UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO


# TÉCNICAS DE EJECUCIÓN COORDINADA DE APLICACIONES PARALELAS BASADAS EN MPI

Autor: David Fernández Sancho

Tutor: David Expósito Singh

Madrid, junio 2017

# Índice de contenidos

# Índice de ilustraciones

# Índice de tablas

*"If something is important enough, even if the odds are against you, you should still do it"*

Elon Musk, Tesla and SpaceX co-founder

# Agradecimientos

En primer lugar quiero agradecer a mi tutor, David Expósito Singh, la oportunidad que me ha brindado para realizar este proyecto y aprender de él. Estoy inmensamente agradecido por la labor que ha realizado durante el desarrollo del mismo y la abundante cantidad de tiempo que me ha dedicado. Más que el título de tutor en este proyecto, debería decirse que ha supuesto un amigo.

A mis padres, por darme la vida y los principios de los que disfruto hoy, por su apoyo incondicional, su ternura, su amor y su ejemplo de inagotable lucha.

A mis hermanas, por ser cómplices de mi vida y confiar en mí.

A mis amigos más cercanos, José, Sarai, María, Mateo y Daniel, por ser lluvia en el desierto, por ofrecer un hombro en el que poder apoyarme y por no decirme lo quiero escuchar sino lo que en realidad es mejor para mí.

A Javier González, por sus enseñanzas, su paciencia, su determinación por ofrecer lo mejor de sí mismo con excelencia y por suponer un punto de referencia en mi vida, que me ha ayudado a no perder la vista de lo verdaderamente importante.

A mi compañero de carrera, Adrían Fernández, por su actitud dispuesta y optimista, por su monumental ayuda y por las noches sin dormir que pasaba a mi lado trabajando.

A mi mejor amigo, Jesús, por su amor desinteresado y por darme el ejemplo de cómo vivir.

Gracias.

# Abstract

## 1 Introduction

In the present stage of the field of computer science, central processing units present certain physical limitations that derive in the need to develop alternative mechanisms to those based on reducing the space occupied by the transistors and increasing the clock frequency. Parallel computing has become one of the main streams to drive progress in this environment.

This section provides an introduction to the project developed. First of all, the motivation for its realization is established. Subsequently, the objectives are presented. Finally, the document structure is described.

### 1.1 Motivation

Traditionally, parallel programming has been used to solve problems that required a high computational cost and that represented the challenges of scientific problems that could be addressed by computer systems. These mechanisms of paralleslism were devoloped by highly specialized proffessionals in the field and in certain very specific scenarios.

At present, the increase in performance achieved by processors, described in Moore's law, begins to present problems due to the physical limits of temperature and space. Added to these disadvantages, are the demands based on solving problems of great complexity and importance on the part of a social and economic environment in continuous technological advance.

For these reasons, alterntive mechanisms are proposed to conserve the benefits provided by the growth of computational performance. Parallel programming has become the main mechanism to achieve this goal, popularized by the aggregation of several independent processing nodes to form computation networks and the development of processors with multiple processing cores.

This popularization results in the development of numerous parallel applications that try to take advantage of all available resources of the systems in wich they are deployed.

In turn, this growing number of parallel applications requires the appearance of various tools that allow an easier and more abstract development, a monitoring of their behaviors and resources consumed, and a management of their execution ajusted to the restrictions and objectives imposed in the system in wich they coexist.

Due to the relevance of the issues presented, the present project proposes the design of a tool capable of centralizing the administration of the parallel applications that are executed in high-performance computer systems and that require a great complexity of processing. These systems have a research or commercial character where optimization of the use of available resources is an issue of crucial importance, so that, through a central administration, numerous techniques can be applied to achieve this goal.

## 1.2 Objectives

FlexMPI is an extension of an implementation of the MPI standard that allows the development of parallel applications providing tools such as modifying the number of processes that make up the application, monitoring its performance data and reconfiguring the application based on constraints or goals imposed by the user. This tool is the result of a PhD thesis carried out at University Carlos III of Madrid.

This project aims to extend the functionalities of FlexMPI, using them to manage the parallel applications of a particular computing system. In order to achieve the main objectives of the project, it is necessary to carry out a series of secondary objectives. Therefore, the objectives pursued with the accomplishment of this work can be listed as:

- **Main objectives:**
  - **Add new features to FlexMPI**. The new functionalities will allow the management of an application from a central controller that will have the role of system scheduler.
  - **Development of an infrastructure for the centralized analysis of the monitored metrics of a parallel application**. This objective establishes the development of a central control component that allows capturing

and analyzing the performance status of all the applications executed in the system.

- **Development of an infrastructure capable of scheduling the parallel applications in a cluster**. The central management component aims to be able to evaluate the workload present in the system and to schedule the applications that must be executed using different efficiency policies.

- **Evaluation of the proposal**. We intend to analyze the results obtained from the development of the exposed mechanisms.

- **Secondary objectives:**

  - **Deployment and analysis of FlexMPI**. This objective will allow to configure the deployment environment and to get used to its functionality.

  - **Development of a set of benchmark applications representative of parallel applications and their integration with FlexMPI.** To represent the different parallel applications that can run in a high-performance computing system, benchmarks will be developed to represent the most common behaviors they may present.

  - **Development of mapping techniques between processes and processing cores**. The objective is to establish mechanisms that allow the allocation of processes of an application to specific processing cores of the system.

  - **Development of a malleability policy**. To make a good use of the resources of the system, the purpose is to develop mechanisms that allow to increase the number of processes associated to a parallel application depending on the state of the system.

  - **Development of an oversubscription policy**. In order to try to increase the volume of work that is produced in the system, the development of techniques that allow several processes to use the same processing core is sought.

## 1.2 Document structure

The structure of the document consists of six differentiated chapters. Below is the purpose of each one.

- **Chapter 1: Introduction.** Presents the motivations for the development of the present project, the objectives that are pursued and the structure of the documentation provided.

- **Chapter 2: State of the art.** Analyzes the current situation of the issues related to this work. It considers the challenges that are presented, the existing alternatives and the work in malleability and parallel scheduling found.

- **Chapter 3: Development Environment.** Exposes the tools studied, considered, and used throughout the project. It offers descriptions of the ones that allow to deepen in their functionalities. It also provides a description of the regulatory framework and the socioeconomic environment associated with the project.

- **Chapter 4: Proposal.** Describes the main project's contribution. It explains the way to proceed to achieve the pursued objectives, the analysis of the system considered and the proposed design.

- **Chapter 5: Evaluation.** Presents validation tests carried out, the environment in which they are executed and tests that allow to evaluate the performance of the different mechanisms designed.
- **Chapter 6: Conclusions and future work.** The document ends by analyzing the conclusions drawn during the development of the project and presents different approaches of research and development that can be derived in later works.

In addition, the following additional contents are included that complement the documentation provided:

- **Appendix A: Acronyms.** A collection of acronyms used during the document is presented.

- **Appendix B: Installation Manual.** Describes the main aspects to get the project installed.

- **Appendix C: User's Manual.** Describes how to use the system and the flow of execution of the tools developed.

## 2 State of art

This chapter describes the state of the field addressed in this paper. Also, challenges encountered by supercomputing are introduced. Among them are the physical limitations that slow down the increase in performance or problems with the characteristics of these systems. Among these problems are those based on primary and secondary memory, where there are difficulties in the consistency, speed of access, concurrency and location of the data. Of particular relevance are the energy efficiency challenges, in which problems appear in the associated costs in the energy necessary to get these systems to reach the established requirements.

The challenges presented lead to the need to create standards and tools that allow building applications in parallel architectures in a simple and efficient way, such as the message passing interface (MPI) and an extension, called FlexMPI, which has more tools for the development of this type of applications.

MPI is a standard that defines syntax and semantics of the functions contained in a message passing library that supports the development of parallel applications. Message passing is a mechanism used in concurrent programming to ensure synchronization and exclusivity between processes.

FlexMPI is an MPI extension, implemented as a library, which allows the user to specify the performance goal to be achieved and the constraints to which the application is subject. In this way, FlexMPI modifies the application by adding or removing processes when it detects that some of the targets are not being met or that some restriction is in effect.

In this project, the development of a central control infrastructure is proposed, that is integrated with the FlexMPI tool, and that allows to manage the parallel applications of a determined high-performance computational system. Among the management mechanisms is proposed the development of scheduling techniques, process malleability and oversubscription in processing cores.

## 3 Development Framework

This chapter presents the different elements and tools considered throughout the development process of this final grade project.

In the field of parallel computing there are two main paradigms that are defined based on the memory architecture they present: shared memory architectures and distributed memory architectures.

OpenMP is oriented to shared memory architectures, in which the different processing units share the same memory and parts of the same application can run on different processing units. MPI aims at distributed memory architectures, which comprise multiple independent units with their own memory, so that an application can be divided to run on different units. However, today MPI can be beneficial for both distributed memory systems and shared memory systems, thanks to the specifications of its latest versions.

Taking into account that the development of this project will be carried out in a distributed memory cluster, the choice of MPI is established as a standard for parallel system development, providing portability due to its cross-platform migration characteristics and availability. Since MPI is a standard, an implementation called MPICH is chosen as library base for FlexMPI, because it is a free, high-performance, highly portable implementation of the MPI standard.

Among all the parallel application performance monitoring platforms considered, PAPI is established as the monitoring tool used to carry out the present project. The PAPI tool specifies a standard application programming interface for accessing the hardware performance counters that are available on most modern microprocessors.

On the other hand, it is established that the benchmark of the Jacobi method will be the parallel application to use in the system, providing a good parallelism response and the ease in behavior modification to obtain benchmarks whose operations are more intensive in processing, memory or communication instructions.

Note that FlexMPI and added functionality store performance monitoring data that does not involve private, sensitive, or sensitive information. However, parallel applications that must be run on the system could work with personal data, whose access is limited to the application itself, being a problem of its own jurisdiction.

From the point of view of the impact that generates FlexMPI along with the added functionalities on its social environment, it is possible to emphasize the benefits that contribute to a better deployment of parallel systems based on distributed memory. The tool provides the ability to have a central control infrastructure capable of coordinating the applications that are to be executed in the system, monitoring its performance metrics and allowing to schedule its execution according to the objectives defined by the user.

Considering the economic aspects, it is very possible that in the future the contributions made by the present project, together with the original FlexMPI functionalities, can be used in other research projects that have economic financing and commercial objectives.

## 4 Proposal

This chapter is intended to describe the project carried out in this final project.

The development methodology followed has been based on the V-model of development, which is considered an extension of the cascade model, in which each stage of development is strictly ordered, so that the beginning of each stage must wait for the completion of the previous one. In the V-model, instead of progressing in a rigorous sequential manner, the verification and maintenance steps are placed in a "V" form, so that if one of the validation phases is not completed correctly, the phases of development must be readjusted following the form that gives name to the model.

In order to explain the different system requirements, the recommendations provided by the IEEE for software requirements specification (SRS) will be followed.

Parallel applications use native and encapsulated MPI functions, through which invocations are made to the FlexMPI library. The FlexMPI monitoring module is used to obtain the different metrics of an application. This module uses the low level interface of the PAPI library to obtain the data through hardware counters.

To make possible the connection between the parallel applications and the controller to be designed, the communication between them is done through FlexMPI. For this purpose a new modular component has been designed in the library that allows communication with the central controller.

The controller consists of a central infrastructure that manages the execution of the different parallel applications in the system. It is in charge of initializing all the necessary resources for the correct execution of the different tasks and of making the pertinent decisions in the management of the applications of the system. Its main function is to schedule the execution of the different applications in the workload, deciding the order in which they are executed and the resources allocated to them. In order to do this, a thread infrastructure that helps to perform various tasks simultaneously is needed.

For scheduling parallel applications, four types of schedulers have been developed: sequential, out of order, malleability and oversubscription.

Sequential scheduling is based on executing parallel applications in the order defined in the workload, so that if an application can not be executed due to a lack of free processing cores, the scheduler stops until another application finishes, to check again if it can be executed.

Out of order scheduling is based on allowing the execution of the workload processes outside the specified order, if there are resources for their execution, thus encouraging the greater use of resources at the present moment in which scheduling takes place.

Out of order scheduling with malleability is primarily based on performing out of order scheduling. Once applied, which means that at that time there are no more applications in the list of applications waiting to be executed whose minimum number of processes can be covered, if there are free resources, the malleability algorithm is performed.

Oversubscription scheduling aims to exploit the principle that two processes running in the same processing core can run individually more slowly, but have a total execution time less than a sequential execution of the two processes themselves.

In addition, three characterization models have been designed to determine the behavior of a parallel application.

The first one is based on the explicit definition of the behaviour by the user.

The second one performs a false execution of the parallel applications defined in the workload, so that during a period of time the application is executed with the simple aim of determining its behavior. Once the application has been characterized, the execution is interrupted, through a control command that is sent to the communication module of FlexMPI, and it begins to characterize another application or begins the execution of the real system if all the applications already have been characterized.

The last one is based on performing the behavior analysis of an application during the actual execution. To do this, when the scheduling starts a new application, it maps over the processing cores without obeying any rules. Once initiated, for a defined period of time in the system, the behavior of the application is analyzed and characterized.

## 5 Evaluation

This chapter evaluates the system developed considering several approaches. The platform on which the various tests of the system have been executed is the Tucán cluster of the University Carlos III of Madrid, which is a distributed memory multiprocessor.

Three types of testing will be performed, involving coding or implementation testing, design, validation of functionality and compliance with requirements or specifications. The information of each test is presented in tabular form, presenting its characteristics in a compact way.

Performance tests will also be performed to evaluate each of the scheduling techniques developed. In order to achieve this objective, four scenarios designed and a scenario with a random set of applications are proposed. The scenarios designed will be established trying to expose the benefits of each scheduling policy to analyze in which cases it might be better to apply each one.

## 6 Conclusions and future work

This final chapter ends the document explaining the conclusions obtained after the completion of the project. In addition, possible future work and research streams are presented, with the objective of further expanding the functionalities of FlexMPI library and developing more central management mechanisms of the parallel applications present in a high-performance computing system.

### 6.1 Conclusions

As mentioned throughout the document, parallel programming has become a very important area to ensure progress in the field of computer science.

During the realization of the present project the design and implementation of mechanisms have been made to manage the parallel applications of a high performance computing system. In order to achieve the objectives, new functionalities have been developed for the FlexMPI tool, a controller component capable of monitoring and scheduling the applications in a high-performance computing system has been developed, FlexMPI and its external libraries have been analyzed and deployed, used in conjunction with parallel applications based on matrix calculation benchmarks, mapping techniques,

scheduling mechanisms, malleability and oversubscription models have been developed and the results obtained have been evaluated.

Through a central control infrastructure, which is able to communicate with the parallel applications, the performance data, that allows to know the state of the applications, is monitored and their order of execution is decided according to the objectives desired. The architecture of this controller is designed so that further progress can be made in the future with new functionalities and management mechanisms.

One of the policies that allow to decide the way in which the applications of the system are executed is the sequential scheduling. This policy aims to respect the order of arrival of the applications in the system for their execution. In this way, it executes applications in strict order according to the free processing cores of the system, stopping the planning if there are no available resources and waiting until the moment in which those resources are released. Although this type of planning may be just right from the point of view of the age and waiting time of the applications, it is not very efficient, as it often wastes resources that other applications could take advantage of.

In this way, out of order scheduling policy is considered, which does not respect the order defined in the workload and allows to find applications to assign in unused processing cores. It can be clearly concluded that this type of policy significantly improves the execution time of the complete application system.

However, in many cases there are also unused cores. Out of order scheduling with malleability aims to correct this situation. This type of scheduling increases the number of processes if there are free resources and there is no other application that requires a number of processes that allow it to be executed. It can be concluded that it is a very adequate policy and that it assures to take advantage of all the resources of the system in each moment. Nevertheless, it can also present extreme scenarios in which the configurations it adopts are not the most appropriate, since, by increasing an application at the present time, can lead to another application available in the future to can not start and be waiting until the malleabled application ends its execution.

The last policy considered is a very interesting scheduling that tries to take advantage of the resources from another approach. It is based on the fact that while an application has periods of inactivity, such as waiting for data accessed in memory, another application can take advantage of such cycles, in which the processing core is idle, to perform its active processing. However, it is a scheduling that can worsen the execution time of the complete system in various scenarios in which the benefits of performing oversubscription do not cover the penalties derived from the context changes of applications to share the

processing core. It can be deduced from the tests carried out that it is a good type of scheduling policy for those cases in which the objective is not that each application can be executed in the shortest possible time, because in fact its individual execution will be slower, but the main goal is to increase the workload per unit of time that the processor performs, since the complete execution time will be faster.

From the student's point of view, it can be ensured that the objectives pursued at the beginning of the project have been achieved. For the realization, the knowledge acquired in the mention of computers has been fundamental, in which it is deepened on aspects of the development of software of systems and operative systems.

It is worth mentioning the numerous difficulties that arose to carry out the deployment of the system in the Tucán cluster of the University Carlos III of Madrid, that required advanced knowledge of certain aspects of the arquitecture. In addition, there were periods of time in which there were availability problems that delayed the development of the project.

In addition, being a research project mainly, there are not clearly defined references that facilitate its construction, and each small step involved a great effort.

In conclusion, it can be stated that the development of the work has been very satisfactory and interesting, and during its implementation many aspects have been learned about the scope studied and the realization of software projects.

## 6.2 Future work

The purpose of this section is to motivate the development of new works that can continue the development of the present project and previous studies in which it is supported.

Firstly, it is important to note that the student will continue to collaborate in the development of new functionalities for FlexMPI, such as the integration of this work with another final project that develops a graphical interface to control the functionalities offered by FlexMPI. The purpose of this integration is to provide a graphical visualization of the performance metrics that the controller receives and to apply the actions that it allows interactively.

On the other hand, although the developed system is designed to establish its functionalities on several nodes of the same system, it was not the objective of the present project and, therefore, this action is not tuned. The tutor and the

student intend to adjust these new activities, resulting in the possibility of carrying out a scientific publication.

As a new contribution to the designed characterization models, the development of a model able to take into account the behavior of an application dynamically is proposed, since normally it does not behave in a constant way, but interleaves phases of different type during its execution. The key to this modeling is based on extracting internal information from the application itself, an action already performed by the controller developed through the hardware counters.

A learning and prediction model is also proposed, which, based on previous experiences, has the ability to determine efficient planning taking into account the resources and applications existing in the system.

Taking into account the FlexMPI tool, which works the parallelism from the approach of system processes, we come up with the development of a functionality that allows to apply a hybrid model with threads. This model can be beneficial in many scenarios, because they provide less needs for routine communication and memory consumption. In addition, a research work is proposed based on exporting the logic that applies FlexMPI to parallel input/output systems, providing mechanisms of parallelism to intensive applications in this type of operations.

Capítulo 1

# Introducción

En la presente etapa de desarrollo de la ciencia de la computación, las unidades centrales de procesamiento presentan ciertas limitaciones físicas para aumentar su rendimiento, que derivan en la necesidad de desarrollar mecanismos alternativos a los basados en reducir el espacio que ocupan los transistores e incrementar la frecuencia de reloj. La computación paralela se ha convertido en una de las corrientes principales para impulsar el progreso en este entorno.

Este capítulo ofrece una introducción al proyecto fin de grado desarrollado. En primer lugar, se establece la motivación de su realización. Posteriormente, se presentan los objetivos planteados. Por último, se describe la estructura que sigue el documento.

## 1.1 Motivación

Tradicionalmente la programación paralela se ha utilizado para la resolución de problemas que presentaban un alto coste computacional y que suponían los retos de los problemas científicos abordables por sistemas computacionales. Estos mecanismos de paralelismo eran desarrollados por profesionales muy especializados en el ámbito y en determinados escenarios muy específicos.

En la actualidad, el incremento de rendimiento del que disfrutaban los procesadores, descrito en la ley de Moore, comienza a presentar problemas debido los límites físicos de potencia disipada y espacio que se manifiestan. A estos inconvenientes se añaden las demandas basadas en resolver problemas de gran complejidad e importancia por parte de un entorno social y económico en continuo avance tecnológico.

En este contexto, se proponen mecanismos alternativos que permitan conservar los beneficios aportados por el crecimiento del rendimiento computacional. La programación paralela se ha convertido en el mecanismo principal para conseguir este objetivo, popularizada por la agregación de varios nodos de procesamiento independientes para formar redes de cálculo y el desarrollo de procesadores con varios núcleos de procesamiento.

Esta popularización deriva en el desarrollo de numerosas aplicaciones paralelas que intentan aprovechar todos los recursos disponibles de los sistemas en los que se despliegan.

A su vez, este creciente parque de aplicaciones paralelas exige la aparición de diversas herramientas que permitan un desarrollo de las mismas más sencillo y abstracto, una monitorización de sus comportamientos y recursos consumidos y una gestión de su ejecución ajustada a las restricciones y objetivos impuestos en el sistema en el que coexisten.

Debido a la relevancia de las cuestiones planteadas, el presente proyecto fin de grado propone el diseño de una herramienta capaz de centralizar la administración de las aplicaciones paralelas que son ejecutadas en sistemas de computación de alto rendimiento. En estas plataformas, el uso eficiente de los recursos hardware disponibles supone una cuestión de crucial importancia, por lo que, a través de una administración central, se pueden aplicar técnicas que permitan alcanzar dicho objetivo.

## 1.2 Objetivos

FlexMPI es una extensión de una implementación del estándar MPI que permite el desarrollo de aplicaciones paralelas aportando herramientas como modificar dinámicamente (en tiempo de ejecución) el número de procesos que componen la aplicación, monitorizar sus datos de rendimiento y reconfigurar la aplicación en base a restricciones u objetivos impuestos por el usuario. Dicha herramienta es el resultado de una tesis doctoral llevada a cabo en la universidad Carlos III de Madrid.

Con el presente proyecto fin de grado se pretende ampliar las funcionalidades de FlexMPI, en vez de en una sola aplicación, para poder gestionar varias aplicaciones paralelas de un determinado sistema computacional. Para llevar a cabo los objetivos del proyecto, es necesario la realización de una serie de objetivos secundarios. Por tanto, los objetivos perseguidos con la realización de este trabajo son los siguientes:

- **Objetivos Primarios:**
  - **Añadir nuevas funcionalidades a FlexMPI.** Las nuevas funcionalidades que se esperan añadir permitirán la gestión de una aplicación desde un controlador central que tendrá el rol de planificador del sistema.
  - **Desarrollo de una infraestructura para el análisis centralizado de las métricas monitorizadas de una aplicación paralela.** Este objetivo establece el desarrollo de un componente de control central que permita capturar y analizar el estado de rendimiento de todas las aplicaciones ejecutadas en el sistema.
  - **Desarrollo de una infraestructura capaz de planificar los siguientes trabajos a ejecutar.** El componente central de gestión pretende tener la capacidad de evaluar la carga de trabajo presente en el sistema y planificar las aplicaciones que se deben ejecutar utilizando distintas políticas de eficiencia.
  - **Evaluación de la propuesta.** Se pretende analizar los resultados obtenidos del desarrollo de los mecanismos expuestos.
- **Objetivos Secundarios:**
  - **Despliegue y análisis de FlexMPI.** Este objetivo permitirá configurar el entorno de despliegue de la misma y familiarizarse con su funcionalidad.
  - **Desarrollo de un conjunto de *benchmarks* representativo de aplicaciones paralelas e integración de los mismos con FlexMPI.** Para representar las diferentes aplicaciones paralelas que pueden ejecutar en un sistema computacional de alto rendimiento se desarrollarán *benchmarks* que representen los comportamientos más habituales que puedan presentar.
  - **Desarrollo de técnicas de mapeo entre procesos y núcleos de procesamiento.** Se persigue el objetivo de establecer mecanismos que permitan la libre asignación de procesos de una aplicación a núcleos de procesamiento específicos del sistema.
  - **Desarrollo de una política de maleabilidad.** Para realizar un buen aprovechamiento de los recursos del sistema se tiene el propósito de desarrollar mecanismos que permitan incrementar el número de procesos asociados a una aplicación paralela en función del estado del sistema.
  - **Desarrollo de una política de sobresuscripción.** Con el objetivo de intentar incrementar el volumen de trabajo que se produce en el sistema se busca el desarrollo de técnicas que permitan a varios procesos compartir un mismo núcleo de procesamiento.

## 1.3 Estructura del documento

La estructura del documento se compone de seis capítulos claramente diferenciados. A continuación, se presenta el propósito de cada uno.

- **Capítulo 1: Introducción.** Presenta las motivaciones que instan al desarrollo del presente proyecto, los objetivos que se persiguen y la estructura de la documentación aportada.
- **Capítulo 2: Estado de la cuestión.** Analiza la situación actual de las cuestiones relacionadas con este trabajo. Considera los retos que se presentan, las alternativas existentes y los trabajos en maleabilidad y planificación paralela encontrados.
- **Capítulo 3: Entorno de desarrollo.** Expone las herramientas estudiadas, consideradas y utilizadas a lo largo del proyecto. Ofrece descripciones de las mismas que permitan profundizar en sus funcionalidades. Además, proporciona la descripción del marco regulador y el entorno socioeconómico asociado al proyecto.
- **Capítulo 4: Propuesta**. Describe el proyecto realizado. Explica la manera de proceder para conseguir los objetivos planteados, el análisis del sistema considerado y el diseño propuesto.
- **Capítulo 5**: **Evaluación.** Presenta las pruebas de verificación llevadas a cabo, el entorno en el que se ejecutan y ensayos que permitan evaluar el rendimiento de los diferentes mecanismos diseñados.
- **Capítulo 6**: **Conclusiones y trabajo futuro.** Finaliza el documento analizando las conclusiones extraídas durante el desarrollo del proyecto y presenta diferentes aproximaciones de investigación y desarrollo que pueden derivarse en posteriores trabajos.

Además, se incluyen los siguientes contenidos adicionales que complementan la documentación aportada:

- **Anexo A: Acrónimos.** Se presenta una colección de acrónimos utilizados durante el documento.
- **Anexo B: Manual de instalación.** Describe los principales aspectos para conseguir la instalación del proyecto.
- **Anexo C: Manual de usuario.** Describe cómo se deben utilizar y el flujo de ejecución de las herramientas desarrolladas.

Capítulo 2

# Estado de la cuestión

En este capítulo se describe el estado del ámbito tratado en este trabajo. Se exponen los últimos avances y situación actual de los temas abordados a lo largo del proyecto, así como una base de conocimiento para desarrollar posteriormente los objetivos descritos en la sección anterior. En primer lugar, se tratarán los desafíos actuales que se presentan en el entorno de la supercomputación. En segundo lugar, se consideran las bibliotecas de desarrollo de sistemas paralelos MPI y FlexMPI. Posteriormente, se realiza un estudio sobre el trabajo planteado recientemente en maleabilidad. Por último, se desarrolla un breve estudio de los planificadores de aplicaciones paralelas considerados hasta el momento.

## 2.1 Desafíos actuales en la supercomputación: Exascale

*Exascale* hace referencia a sistemas computacionales que tienen la capacidad de mejorar mil veces la capacidad de un equipo supercomputador establecido actualmente, *petascale* [1], y que, por tanto, pueden realizar un mínimo de $10^{18}$ cálculos en coma flotante por segundo, es decir un *exaflops*, que supone mil *petaflops* [2]. En la actualidad, el aumento de rendimiento basado en el incremento del número de transistores de los sistemas computacionales, descrito en la ley de Moore, comienza a presentar numerosos problemas, como el consumo de energía y las temperaturas que pueden alcanzar los chips desarrollados o el coste que supone reducir el espacio entre transistores en un solo nanómetro [3]. Es por este motivo, que el objetivo que se persigue en la actualidad es aumentar la capacidad de cálculo de estos sistemas a través de mecanismos de computación paralela, como la inclusión de *cores* en los microprocesadores o aumentar la cantidad de *threads* que puede admitir un solo

*core*. A su vez, la búsqueda de este objetivo desencadena en que se estén introduciendo cambios en las arquitecturas paralelas tradicionales. Se espera que dentro de esta y la siguiente década, el rendimiento provocado por el número de *cores* en un solo microprocesador rivalizará con el del número de nodos establecidos en las arquitecturas más clásicas de supercomputación paralela, lo que deriva en la necesidad de adaptar estas arquitecturas y sus algoritmos a medida que estos evolucionan [4].

Los retos que enfrentan los sistemas con características *exascale* son numerosos y variados. Se centran, principalmente, en memoria primaria y secundaria, donde se presentan problemas de consistencia, velocidad de acceso, concurrencia y localidad de los datos. Resultan de especial relevancia los retos de eficiencia energética, en los que aparecen problemas en los costes asociados en la energía necesaria para conseguir que estos sistemas lleguen a los requisitos establecidos [5].

La estrategia clave para progresar en el desarrollo de estos sistemas viene dada por la cooperación entre el diseño software y el diseño hardware. La primera comunidad debe hacer uso de las tecnologías más recientes y punteras para desarrollar un código eficiente, mientras que, la segunda comunidad, debe hacer lo posible por aumentar el rendimiento de las aplicaciones y resolver sus problemas más comunes desarrollando tecnologías que las apoyen y aportando la documentación necesaria y pertinente para su fácil utilización [4].

Los retos expuestos conducen a la necesidad de crear estándares y herramientas que permitan construir aplicaciones en las arquitecturas paralelas de manera sencilla y eficiente, como la interfaz de paso de mensajes (MPI) y una extensión de la misma denominada FlexMPI, que cuenta con más herramientas para el desarrollo de este tipo de aplicaciones.

## 2.2 MPI y FlexMPI: soporte para la maleabilidad

Debido a los retos considerados en los sistemas de computación de alto rendimiento (HPC), los mecanismos de maleabilidad resultan un ámbito de especial interés. La maleabilidad computacional es la capacidad de una aplicación de expandir dinámicamente su número de procesos, repartiendo la carga de trabajo para ejecutarla paralelamente. Esta cualidad es realmente útil en sistemas HPC debido a que permite que las aplicaciones se expandan para ejecutarse más rápidamente en función de los recursos disponibles en el sistema, dados por la carga de trabajo existente en ese momento. De esta manera, una

aplicación paralela puede ejecutarse más rápida o lentamente en función del número de aplicaciones que compartan los recursos con ella [6].

MPI es un estándar que define sintaxis y semántica de las funciones contenidas en una biblioteca de paso de mensajes que sirve de apoyo para desarrollar aplicaciones paralelas. El paso de mensajes supone un mecanismo utilizado en programación concurrente para asegurar sincronización y exclusividad entre procesos.

Está implementado para los lenguajes de programación *C y Fortran*, lenguajes que no incluyen por defecto herramientas que faciliten los paradigmas más comunes en la creación de este tipo de aplicaciones. Con anterioridad al desarrollo de MPI, los desarrolladores realizaban aplicaciones no portables fuertemente dependientes de las plataformas en las que se ejecutaban. Y precisamente, el estándar MPI trata de solucionar estos problemas. Esta biblioteca está diseñada para permitir a los desarrolladores crear programas que puedan ejecutarse en la mayoría de arquitecturas paralelas. En su diseño colaboraron compañías de prestigio, como IBM, Intel, TMC, etc., y especialistas en aplicaciones y bibliotecas paralelas, como PVM, Linda, etc.

Además, una característica sobresaliente es que **soporta** la ejecución de aplicaciones distribuidas en **hardware heterogéneo**, contando con diferentes plataformas trabajando en un mismo problema.

Sus fundamentos se basan en que, en su diseño convencional, el número de procesos que utilizará la aplicación se define antes de la ejecución y **no se crean procesos adicionales** durante su actuación. Cada proceso se identifica unívocamente y se pueden agrupar en colecciones de procesos que pueden enviarse mensajes entre sí. Las llamadas de la biblioteca MPI se puede clasificar en 4 tipos: utilizadas para inicializar, administrar y finalizar las comunicaciones, empleadas para transferir datos entre un par de procesos, dedicadas a transferir datos entre un conjunto de procesos y utilizadas para crear tipos de datos definidos por el usuario [7].

Por estas características, la mayoría de aplicaciones científicas que se ejecutan en *clusters* HPC están implementadas utilizando MPI. Sin embargo, la mayor parte de estas implementaciones resultan en un paralelismo rígido donde el número de procesos asignados está fijo durante la ejecución del programa.

Con el objetivo de proporcionar a las aplicaciones paralelas de maleabilidad a través de una reconfiguración dinámica de rendimiento, nace FlexMPI. FlexMPI, supone una extensión de MPI, implementada como una biblioteca, que permite al usuario especificar el objetivo de rendimiento a conseguir y las restricciones a las que se sujeta la aplicación. De esta manera, FlexMPI modifica la aplicación

añadiendo o eliminando procesos cuando detecta que algunos de los objetivos no se están cumpliendo o que alguna restricción está vigente.

Sus fundamentos se basan en ser una biblioteca eficiente que soporta maleabilidad, balanceo de carga, redistribución de datos, reconfiguración dinámica de las funcionalidades y monitorización sobre las aplicaciones que la implementan. Desarrolla un modelo de predicción que puede estimar el rendimiento de la aplicación paralela. Ofrece un API que da acceso a las funcionalidades de FlexMPI desde cualquier aplicación MPI.

Además, aporta notables contribuciones como una monitorización automática vía contadores hardware de rendimiento, predicción del rendimiento futuro y una reconfiguración dinámica basada en restricciones de costes y eficiencia energética, todo ello, aplicable sobre aplicaciones paralelas que ya implementan MPI requiriendo un esfuerzo de desarrollo mínimo [8].

## 2.3 Trabajo reciente en maleabilidad

En el presente apartado se realiza un breve análisis sobre los trabajos actuales relacionados con el ámbito de la maleabilidad.

Cabe destacar, que existen dos aproximaciones para maleabilizar aplicaciones paralelas: maleabilidad estática y dinámica. En la primera aproximación es necesario parar la aplicación paralela durante su ejecución, para reconfigurarla en base a los datos recogidos sobre la ejecución realizada y volverla a ejecutar aplicando la maleabilidad pertinente [9]. Sin embargo, la maleabilidad dinámica permite la reconfiguración dinámica de la aplicación mientras se está ejecutando, sin necesidad de detenerla. La maleabilidad dinámica proporciona mayores beneficios en todos los aspectos de la ejecución, aunque requiere un diseño más complejo, y por este motivo, se considerarán solo los trabajos enfocados sobre esta aproximación.

En [10], se hace uso de una interfaz de gestión de procesos, semejante a la que utiliza FlexMPI, para permitir maleabilidad a aplicaciones MPI iterativas mediante mapeo dinámico en CPU. Sin embargo, en la monitorización llevada a cabo no se considera el rendimiento de tiempo de ejecución de la aplicación, lo que se traduce en el uso ineficiente de los recursos disponibles. Este inconveniente no sucede en FlexMPI.

En [11], se presenta un *framework* prototipo llamado ReSHAPE, capaz de soportar maleabilidad dinámica de aplicaciones MPI iterativas ejecutadas en plataformas de memoria distribuida. Incluye un planificador que admite el redimensionamiento de las aplicaciones, un API para interactuar con el

planificador y una biblioteca que hace viable el cambio de tamaño. Intenta realizar una optimización basada en intentos. En primer lugar, incrementa el número de procesos de la aplicación. Si el cambio ha producido mejoras en el tiempo de la iteración de la aplicación, realizará otro intento incrementando más la aplicación. Si el tiempo de iteración es peor tras el cambio, reducirá el número de procesos. En comparación, FlexMPI presenta un sistema de predicción mucho más sofisticado, dónde se analizan los datos recogidos de contadores hardware de rendimiento y se estima el rendimiento que tendría la aplicación antes de realizar el cambio. Por otra parte, ReSHAPE asume que los tiempos de iteración de la aplicación y las operaciones de comunicación son siempre iguales entre sí, mientras que FlexMPI tiene en cuenta que el patrón de estos parámetros podría cambiar y se basa en información obtenida de varias iteraciones anteriores.

En [12], se presenta una solución global para ofrecer maleabilidad virtual aplicando una estrategia de asignación de procesadores bautizada como *Folding by JobType.* Dicha estrategia se apoya en los conceptos de doblado y maleabilidad de procesos, decidiendo el número de procesos óptimo en el momento inicial de ejecución y cuándo duplicar el trabajo aprovechando la disponibilidad de los procesadores.

En [13] y [14], se presenta una extensión de la biblioteca PCM, que supone una biblioteca a nivel de usuario para aplicaciones MPI iterativas. Dicha extensión es capaz de permitir a las aplicaciones la reconfiguración de acciones de dividir la aplicación en procesos o mezclarlos en un solo proceso, además de dar la posibilidad de migración de procesos, que tiene en cuenta la localidad de los mismos. Su principal inconveniente es que necesita añadir un gran número de primitivas en el código fuente, perjudicando el desarrollo de las aplicaciones paralelas.

## 2.4 Planificación de aplicaciones paralelas

En un sistema de computación de alto rendimiento en el que se ejecutan varias aplicaciones paralelas que persiguen ser ejecutadas en el menor tiempo posible, resulta de gran interés los mecanismos de planificación que deciden qué aplicaciones se ejecutan y en qué orden. A continuación, se consideran estudios que aportan contribuciones al ámbito de la planificación de aplicaciones paralelas para establecer bases en el desarrollo del presente trabajo.

Los sistemas *cluster* con planificación *batch,* normalmente, solo admiten una asignación estática de los recursos para las aplicaciones antes de comenzar a ejecutarlas. Después de su comienzo, no admiten ningún tipo de maleabilidad,

aunque se trate de aplicaciones que evolucionen impredeciblemente a lo largo de la ejecución. Esta situación deriva a que los usuarios que desean ejecutar aplicaciones en el sistema realicen una asignación estática de una gran cantidad de recursos para tener en cuenta partes en las que se haga uso de una gran demanda de los mismos. Dichos comportamientos se traducen en el desperdicio de los recursos cuando las aplicaciones realmente no los necesitan. En [15], se propone un sistema *batch* con facilidades dinámicas de asignación de recursos basados en demanda, que proporciona una asignación justa entre aplicaciones rígidas y aplicaciones que evolucionan en el sistema. Destaca el uso de un mecanismo de relleno, en el que aquellos trabajos de baja prioridad se ejecutan en nodos que no están siendo usados y proporciona funcionalidades interesantes como una interfaz a través de la cual las aplicaciones pueden demandar o liberar recursos en tiempo de ejecución, encolar peticiones dinámicamente para su planificación en el servidor del sistema o la capacidad de asociar y desasociar nodos y trabajos dinámicamente durante el tiempo de ejecución.

En [16], se propone una nueva estrategia de trabajo maleable que es capaz de gestionar más eficientemente trabajos rígidos, maleables y que pueden evolucionar impredeciblemente. De la misma manera, destaca un mecanismo en el que los trabajos de menor prioridad se ejecutan fuera de orden en nodos de computación que no están siendo usados.

En [17], se presentan dos metodologías distintas para la planificación de aplicaciones MPI maleables. La primera consiste en utilizar puntos clave y reinicio y la biblioteca SCR, mientras que la segunda utiliza una redistribución dinámica de datos y el uso de la ULFM API. Esta interfaz incluye soporte para tolerancia a fallos en el estándar MPI. Añade códigos de error que el desarrollador de aplicaciones puede utilizar para depuración de fallos y funciones para indicar a la aplicación qué hacer en caso de encontrar un error o un comportamiento inesperado. Esta agregación de funcionalidades es especialmente relevante debido a que la mayoría de implementaciones basadas en MPI no incluyen patrones de recuperación ante fallos de nodos en la arquitectura en tiempo de ejecución. Su objetivo es el de la convivencia de la computación de alto rendimiento y *Big Data*.

En [18], se presenta con más detalle el planificador del *framework* ReSHAPE comentado anteriormente. Dicho planificador es capaz de soportar redimensionamiento de las aplicaciones paralelas dinámicamente, ante los planificadores estáticos actuales. Utilizan descriptores del rendimiento basados el tiempo de uso de CPU y el tiempo de distribución de datos por parte de las aplicaciones. Sin embargo, al contrario que FlexMPI, no utiliza contadores hardware ni mide tiempos de operaciones I/O. Proponen varías políticas de planificación basadas en diversos criterios: máximo beneficio global, causar

menor impacto en el sistema, contrato justo entre aplicaciones de distinto tipo, favorecer aplicaciones en la cola de espera sobre las aplicaciones actualmente en ejecución en el sistema y viceversa. Utilizan el conjunto de *benchmarks* NAS, que se focalizan en la evaluación de productividad de supercomputadores de alto rendimiento, los cuales han sido desarrollados por la División de supercomputación avanzada de la NASA.

En [19], ante las estrategias de planificación en sistemas paralelos que favorecen a los trabajos cortos dejando de lado el gran desempeño de los trabajos largos, se propone una estrategia de planificación que maximiza la utilización de recursos, mejorando el rendimiento al permitir que las aplicaciones se adapten a las variaciones de carga. Las evaluaciones realizadas suponen cargas de trabajo reales demostrando que dicha política alcanza mejores resultados que las tradicionales políticas de relleno de trabajos, especialmente en cargas medias y altas del sistema. Cabe destacar, que no se enfoca en la maleabilidad, puesto que no es utilizada durante la planificación, simplemente se basa en la migración de procesos.

En [20], se realiza un estudio de la combinación de las medidas de rendimiento y la prioridad de las aplicaciones asignadas por los usuarios para informar sobre las decisiones sobre la expansión y demanda de un conjunto de procesadores específico para la realización de un determinado trabajo.

Por último, en [21], se presenta el planificador *multicluster* de políticas de maleabilidad *Koala* con la ayuda del *framework Dynaco* para la maleabilidad de aplicaciones. Se trata de planificador diseñado especialmente para sistemas *multicluster*. Aplica principalmente dos tipos de políticas: la política FPSMA, que favorece a las aplicaciones maleables iniciadas con anterioridad, y la política EGS, que intenta balancear los procesadores disponibles ante las aplicaciones maleables.

Una vez analizadas las principales cuestiones relacionadas con el desarrollo de este trabajo fin de grado, se propone el desarrollo de una infraestructura de control central que esté integrada con la herramienta FlexMPI, y que permita administrar las aplicaciones paralelas de un determinado sistema computacional de alto rendimiento. Entre los mecanismos de administración se propone el desarrollo de técnicas de planificación, de maleabilidad de procesos y de sobresuscripción en núcleos de procesamiento.

Capítulo 3

# Entorno de desarrollo

En este capítulo se presentan los diferentes elementos y herramientas consideradas a lo largo del proceso de desarrollo de este trabajo fin de grado. Todo componente utilizado o evaluado, desde herramientas de desarrollo software, sistemas operativos o librerías externas, es expuesto en esta sección. En primer lugar, se presenta un estudio de las características de las herramientas de desarrollo de aplicaciones paralelas consideradas. En segundo lugar, se exhiben las diferentes plataformas de monitorización de rendimiento que se tuvieron en cuenta para obtener datos de la ejecución de las aplicaciones paralelas. Posteriormente, se ofrece una descripción de los *benchmarks* utilizados en representación de las aplicaciones paralelas del sistema. Acto seguido, se presentan las diferentes herramientas utilizadas durante el desarrollo. Por último, se describen los marcos de regulación y el entorno socio-económico por los que se rige el presente proyecto.

## 3.1 Alternativas software para el desarrollo de aplicaciones paralelas

### 3.1.1 OpenMP y MPI

En el ámbito de la computación paralela existen dos paradigmas principales que se definen en base a la arquitectura de memoria que presentan: arquitecturas de memoria compartida y arquitecturas de memoria distribuida. La Ilustración 1 y la Ilustración 2 presentan un esquema de los dos paradigmas descritos.

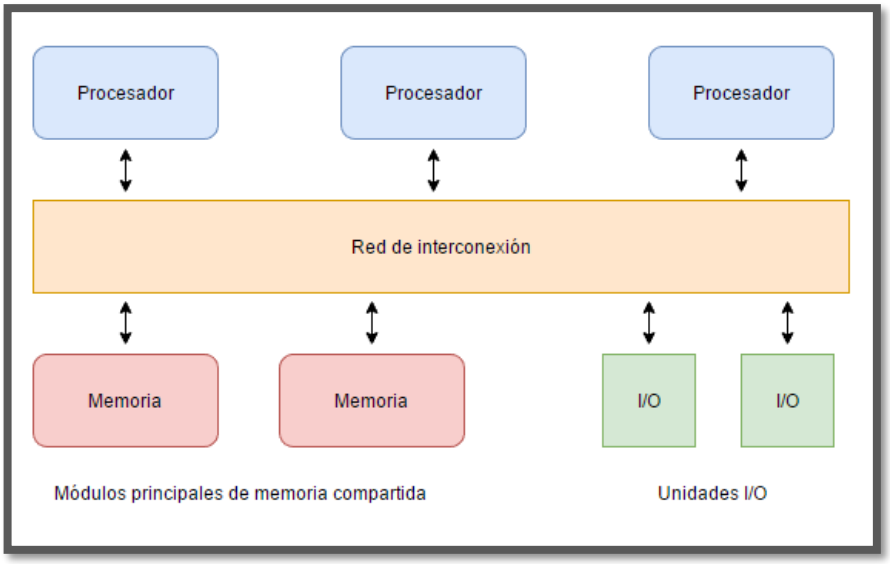



*Ilustración 1. Arquitectura de memoria compartida*

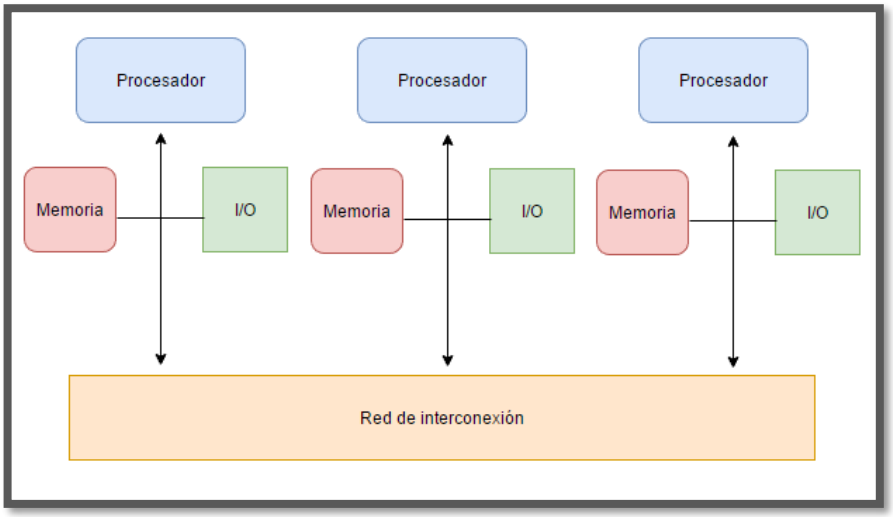



*Ilustración 2. Arquitectura de memoria distribuida*

Las arquitecturas de memoria compartida son aquellas en las que los diferentes procesadores son capaces de acceder a toda la memoria del sistema disponible. Presentan una situación en la que cada nodo de procesamiento comparte tablas de páginas, memoria virtual y todo aquello que se aloja en memoria principal con otros nodos en la misma red de interconexión. Cualquier procesador puede acceder a los mismos datos, así como cualquier dispositivo de entrada/salida [22]. Por otra parte, en las arquitecturas de memoria distribuida cada procesador solo es capaz de acceder a una porción de memoria limitada, disponible para él y no para otros procesadores. Cada unidad de procesamiento dispone de su propia memoria privada, en la que las tareas computacionales sólo pueden operar con datos locales, y en el caso de tener que operar con datos remotos,

estas tareas deben comunicarse con otras unidades de procesamiento remotas [23]. Los procesadores no tienen información de dónde residen todos los datos de las memorias remotas, por lo que deben solicitarlos explícitamente mediante un protocolo de comunicaciones. Además, se enfrentan a la posibilidad de penalizaciones en el rendimiento para acceder a los datos remotos.

Los sistemas de memoria distribuida no presentan problemas de condiciones de carrera y mejoran la escalabilidad. Por su parte, los sistemas de memoria compartida cuentan con la ventaja de disponer de un espacio de memoria unificado, lo que permite que los datos sean accedidos con una baja latencia, en lugar de acceder mediante el intercambio de grandes cantidades de datos entre distintas unidades de procesamiento.

*OpenMP* se orienta a arquitecturas de memoria compartida, en la que las distintas unidades de procesamiento comparten la misma memoria y partes de la misma aplicación pueden ejecutarse en una unidad distinta. Por su parte, MPI pone su objetivo en las arquitecturas de memoria distribuida, que comprenden múltiples unidades independientes con su propia memoria, por lo que una aplicación se puede dividir para ejecutarse en unidades distintas. Sin embargo, actualmente MPI puede resultar beneficioso tanto para los sistemas de memoria distribuida como para los sistemas de memoria compartida, gracias a las especificaciones de sus últimas versiones.

*OpenMP* es un conjunto de directivas de compilación y rutinas de librerías de tiempo de ejecución que son utilizadas para especificar paralelismo en **sistemas de memoria compartida** para los lenguajes de programación *Fortran* y *C*. Su uso resulta sencillo puesto que involucra la inclusión de las directivas *pragma* en el código. Se basa en una aproximación de programación fundamentada en *threads*, en la que a partir de un solo proceso se pueden derivar tantos *threads* como se requiera [24]. En la Ilustración 3 se presenta este modelo de diseño paralelo de manera esquemática, denominado *fork-join,* que utiliza *OpenMP.*

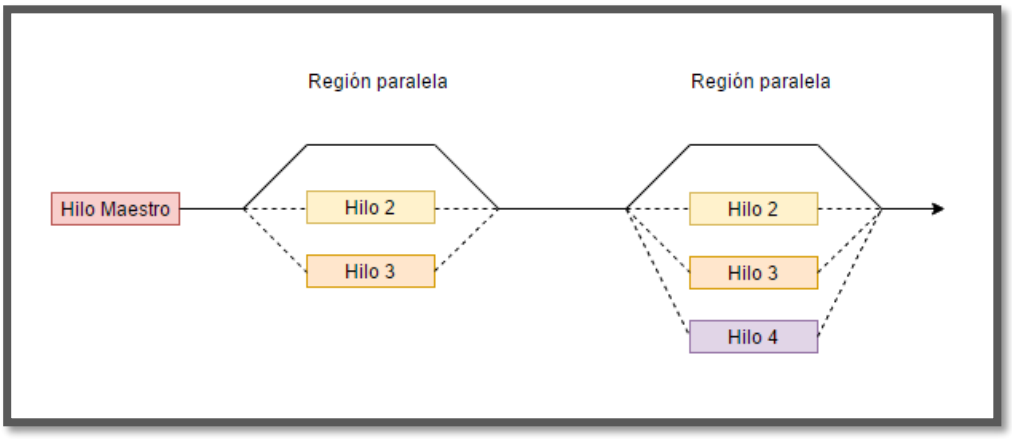



*Ilustración 3. Modelo fork-join*

El modelo *fork-join* es un mecanismo de construcción de aplicaciones paralelas en el que la ejecución se ramifica en puntos designados del programa, que pueden aprovechar las ventajas de realizarse paralelamente, para unirse en un punto posterior y reanudar la ejecución secuencial.

MPI, como se ha explicado en el anterior capítulo, es un estándar que define sintaxis y semántica de las funciones contenidas en una biblioteca de paso de mensajes que sirve de apoyo para desarrollar aplicaciones paralelas en los lenguajes de programación *Fortran* y *C*. Aunque convencionalmente sólo se basaba en programación fundamentada en procesos, en sus últimas versiones admite también una aproximación basada en *threads*. En comparación con *OpenMP*, MPI requiere un mayor esfuerzo de implementación y necesita ajustarse adecuadamente para funcionar óptimamente con los recursos disponibles.

Realizando un análisis más exhaustivo se pueden determinar sus características más diferenciadoras.

En primer lugar, MPI cuenta con múltiples versiones e implementaciones que pueden ser compiladas en multitud de plataformas, mientras que *OpenMP*, es fuertemente dependiente del compilador por lo que su implementación es difícil de modificar.

En cuanto al sistema objetivo, como se ha comentado anteriormente, MPI puede abarcar arquitecturas de memoria compartidas y distribuidas, además de utilizar aproximaciones basadas en procesos o *threads*, mientras que *OpenMP* solo se puede focalizar en sistemas de memoria compartida que utilicen aproximaciones basadas en *threads.*

Por otra parte, MPI posee sobrecarga en su ejecución asociada a las transferencias de datos y mensajes, cuando en *OpenMP* no existe dicha sobrecarga, ya que los hilos pueden compartir datos y variables.

Sin embargo, *OpenMP* sí que presenta sobrecarga asociada a condiciones de carrera, siendo estos problemas inherentes a su modelo de compartición.

Teniendo en cuenta el esfuerzo de desarrollo, MPI requiere la realización de numerosos cambios para obtener una versión paralela de una aplicación a partir de su versión secuencial y puede ser más difícil de depurar, mientras que el código en *OpenMP* es más sencillo de implementar y de mantener.

En [27], se presentan mecanismos para automatizar una traducción de aplicaciones de memoria compartida *OpenMP* a aplicaciones de paso de mensajes MPI para su ejecución en sistemas de memoria distribuida, con el objetivo de extender la facilidad de creación de aplicaciones *OpenMP* a una variedad más amplia de plataformas.

Además, cabe destacar la existencia de soluciones híbridas que intentan combinar las ventajas de ambas aproximaciones, como por ejemplo evitar el paso de mensajes entre procesos dentro de un mismo nodo del sistema, ya que pueden hacer uso de las ventajas asociadas a la compartición de memoria disponible [26].

En [25], se realiza un estudio para comparar el rendimiento entre programación en memoria compartida y mecanismos de paso de mensajes, utilizando *Java* y *C*, *OpenMP*, para memoria compartida y MPI, como mecanismo de paso de mensajes. Las conclusiones que se obtienen resaltan por la poca diferencia existente entre el rendimiento de los dos sistemas, el cual depende del software y hardware utilizado. Por lo cual, no se puede establecer que un sistema sea mejor que otro, y la elección estará subordinada a la arquitectura subyacente del entorno de desarrollo.

En conclusión, teniendo en cuenta que el desarrollo del presente proyecto se realizará en un *cluster* de memoria distribuida, se establece la elección de MPI como estándar para el desarrollo del sistema paralelo, proporcionando portabilidad, debido a sus características de migración entre plataformas, y disponibilidad, al consistir de un estándar de dominio público con numerosas implementaciones.

### 3.1.2 Versiones de MPI

Puesto que MPI es solo un interfaz, dispone de diferentes implementaciones realizadas por numerosos vendedores. El código fuente de dichas versiones es compatible entre sí dando soporte a arquitecturas heterogéneas. A continuación, se presentan las versiones más relevantes.

- **MPICH** es una implementación gratuita, de alto rendimiento y altamente portable del estándar MPI. Sus principales objetivos son el de proveer una implementación de MPI eficiente que soporte numerosas plataformas de computación y comunicación, incluyendo *clusters* de servicios básicos, redes de alta velocidad y sistemas de computación de alta gama, y el de contar con un entorno modular fácil de ampliar para otras implementaciones derivadas. Es cómplice de todas las versiones del estándar MPI, y se distribuye para una gran cantidad de sistemas operativos tipo *Unix/Linux* y *Windows*.

- **OpenMPI** es una implementación *open source* del estándar MPI desarrollada y mantenida por un consorcio de asociaciones académicas, de

investigación y de industria. El principal objetivo de este proyecto es el de combinar la experiencia, recursos y tecnología de la comunidad de computación de alto rendimiento para intentar construir la mejor biblioteca de MPI disponible. Busca proporcionar un rendimiento extremadamente alto y competitivo, proporcionando una biblioteca estable para la investigación pública y el desarrollo comercial.

– **MPI/Pro** es una implementación comercial del estándar MPI. Se propone optimizar la construcción de aplicaciones de procesamiento paralelo. Es soportado en una amplia variedad de sistemas operativos, incluyendo *Linux* y sistemas operativos en tiempo real. Sus desarrolladores fueron los coautores de las dos primeras versiones del estándar MPI, así como de la primera versión de la implementación gratuita MPICH.

Para el desarrollo del presente proyecto solo se han considerado implementaciones gratuitas. Las más relevantes son MPICH y OpenMPI. Cada una está diseñada para satisfacer diferentes necesidades, sin embargo, MPICH es la que aporta una mayor cobertura del estándar MPI para diferentes plataformas hardware, implementando cada una de las versiones del estándar. OpenMPI recientemente ha dado soporte a la última versión del estándar, y algunas de sus funcionalidades presentan fallos de implementación.

Por las razones presentadas, se va a establecer MPICH como la implementación del estándar de MPI a ser utilizada en el desarrollo del presente trabajo.

## 3.2 Plataformas de monitorización de rendimiento de aplicaciones paralelas

Para hacer un uso eficaz de los recursos disponibles en los sistemas de computación paralela existen herramientas que incrementan el nivel de paralelismo con funcionalidades tales como evaluación del rendimiento, depuración y control de una aplicación en tiempo de ejecución. De esta manera, a partir de la información que aportan estas plataformas es posible explotar al máximo la arquitectura en la que se emplea computación paralela. Algunas de estas herramientas también son capaces de mostrar fallos que se producen en el rendimiento, en las comunicaciones entre procesos o en el comportamiento de la aplicación [28].

A continuación, se exponen las herramientas examinadas con el fin de justificar la elección de aquella que más se ajusta a las necesidades del presente trabajo.

### 3.2.1 Intel VTune Amplifier

*Vtune Amplifier* es una herramienta de análisis de rendimiento, principalmente para código *C*, *C++* y *Fortran*, que puede identificar en qué se consume el tiempo de una aplicación durante su ejecución. Concretamente, en aplicaciones paralelas, como aplicaciones basadas en MPI o en OpenMPI, puede determinar el grado de concurrencia y detectar cuellos de botella causados por primitivas de sincronización. Proporciona configuraciones de análisis predefinidas, denominadas *experimentos*, con el objetivo de abordar diferentes cuestiones acerca del rendimiento.

Entre algunas de sus otras numerosas funcionalidades permite localizar aquellas funciones en el código que emplean mayor uso de CPU, analizar el ancho de banda de acceso a memoria y la predicción de fallos, determinar los tiempos de bloqueo y espera, muestrear la pila de la aplicación y los eventos hardware y realizar un análisis del almacenamiento empleado.

La Ilustración 4 presenta la interfaz de usuario gráfica de la herramienta *Vtune Amplifier*.

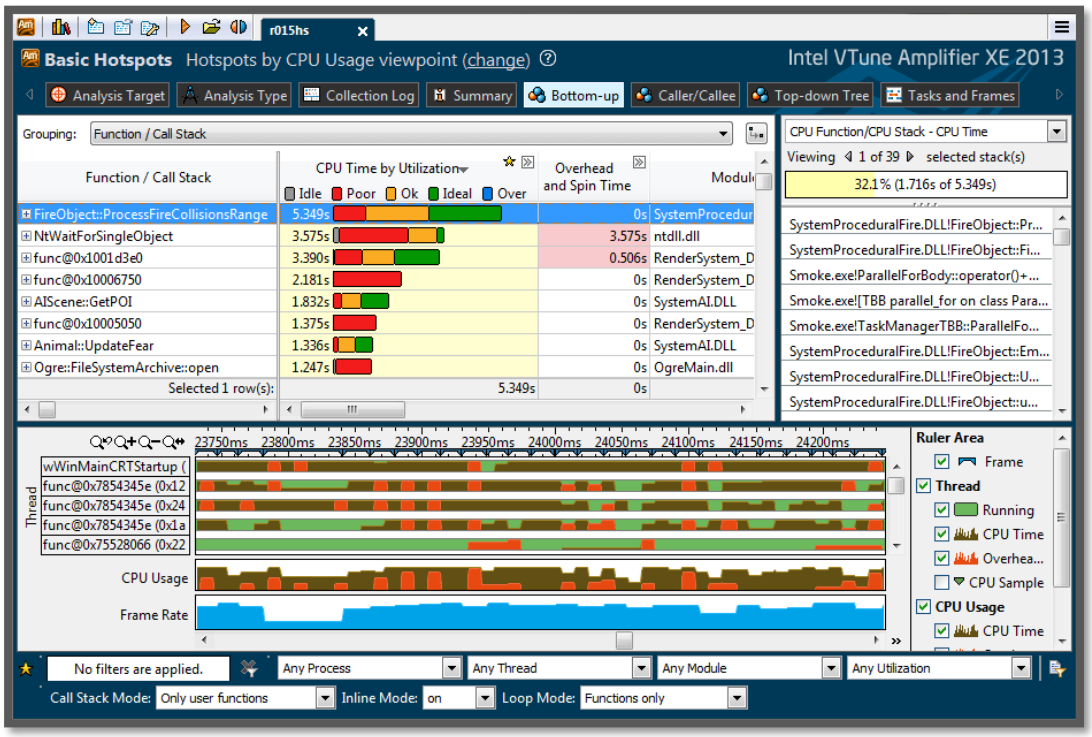



*Ilustración 4. Interfaz de la herramienta Intel VTune Amplifier*

### 3.2.2 Vampir

*Vampir* es un entorno de desarrollo para el análisis de rendimiento que permite a los desarrolladores comprender rápidamente el comportamiento de una aplicación con un nivel de detalle preciso. Es una herramienta que genera trazas de ejecución de subrutinas y bloques de código de aplicaciones paralelas con MPI. Emplea la biblioteca *VampirTrace* para realizar la monitorización de todas las comunicaciones. Los datos obtenidos ante una ejecución de un programa paralelo pueden ser analizados mediante una colección de vistas gráficas especializadas, lo cual constituye su característica más importante y única. Sus representaciones gráficas son interactivas (soportan navegación y zoom) e intuitivas, permitiendo la visualización de las métricas en gráficas de líneas de tiempo, de barras, tarta e histogramas. Además, posee funcionalidades de búsqueda y filtrado para permitir identificar rápidamente cuellos de botella o anomalías en los datos registrados [29].

Es una herramienta comercial y se distribuye en versiones distintas, con diferentes funcionalidades, para satisfacer distintas necesidades y objetivos. La Ilustración 5 presenta una de las vistas gráficas que genera *Vampir*.

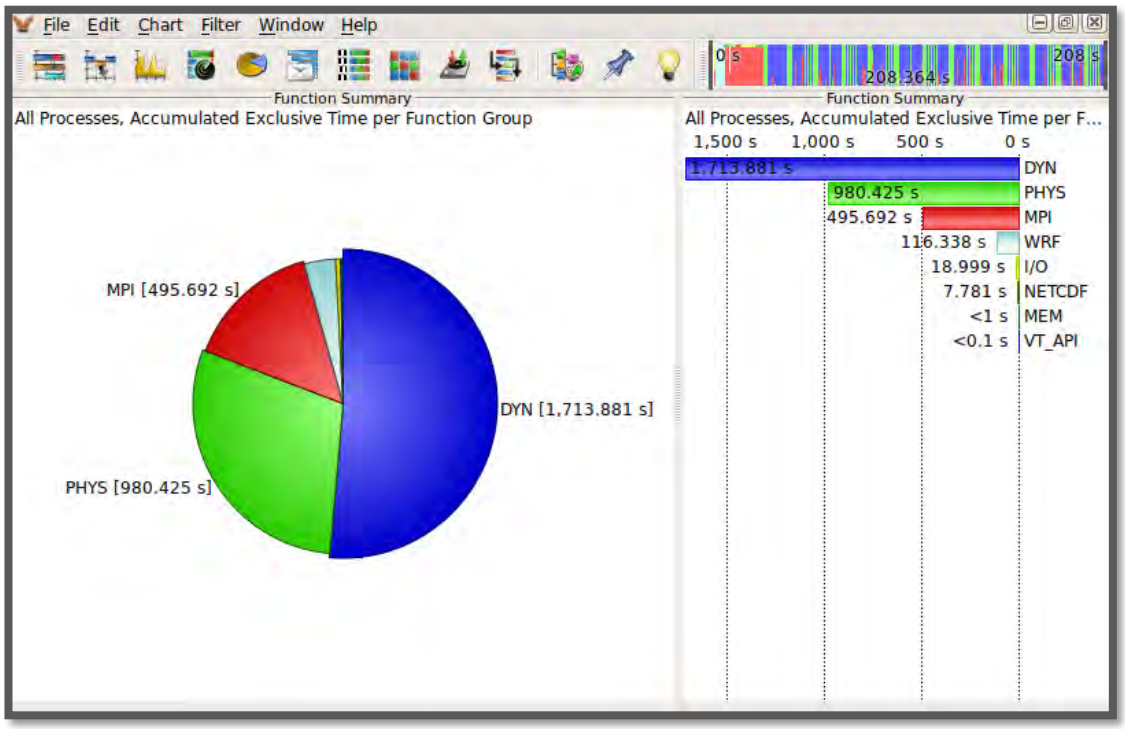



*Ilustración 5. Interfaz de usuario gráfica de Vampir*

### 3.2.3 Periscope

*Periscope* es una herramienta escalable de análisis de rendimiento automático. Permite a los desarrolladores identificar cuellos de botella en sus aplicaciones, debido a que desarrolla hipótesis de manera automática que comúnmente cumplen los comportamientos que generan cuellos de botella y verifica en tiempo de ejecución. Utilizando los diversos *plugins* de ajuste que incluye es posible encontrar automáticamente la combinación más apropiada de las configuraciones de la aplicación, tales como los *flags* de compilación, ajustes MPI y el número de *threads* que *OpenMP* debe generar durante las regiones paralelas.

Se compone de un *frontend* y de una jerarquía de agentes de comunicación y análisis. Dichos agentes buscan de forma autónoma ineficiencias en los procesos que ejecuta la aplicación.

Los procesos de la aplicación están vinculados con un sistema de monitorización que ofrece una interfaz de peticiones de monitorización (MRI). Los agentes se conectan al monitor a través de *sockets*. La interfaz MRI permite a los agentes configurar las medidas, iniciar o detener la ejecución u obtener datos.

La aplicación y el conjunto de agentes se inician a través del proceso *frontend*. Éste analiza el conjunto de procesadores del sistema que se encuentran disponibles y determina la asignación de los procesos de la aplicación y de los agentes de análisis e inicia la aplicación y la jerarquía de agentes. Después del arranque, se envía un comando a los agentes de análisis para empezar una búsqueda para detectar las propiedades de rendimiento pertinentes y son reportadas de vuelta al proceso *frontend.*

*Periscope* puede ser utilizado también para trabajos *batch.* La Ilustración 6 muestra los resultados de la ejecución de la herramienta.

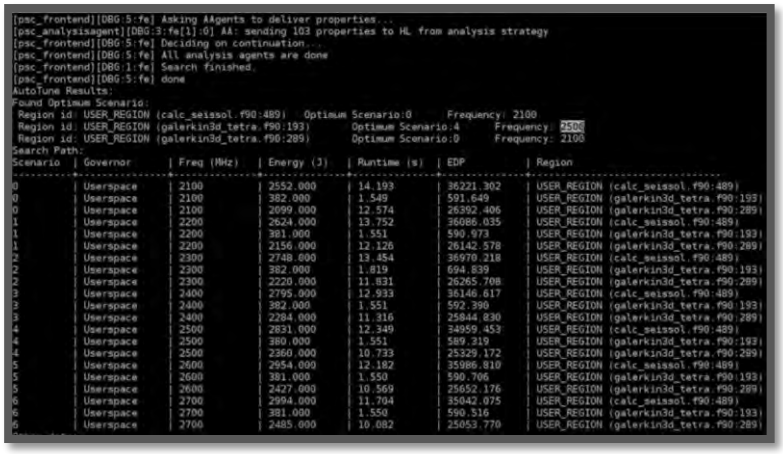



*Ilustración 6. Salida estándar de periscope*

### 3.2.4 TAU

TAU es un sistema de análisis de rendimiento paralelo diseñado como un *framework* robusto, flexible, portable e integrado que dispone de herramientas para instrumentación, medición, análisis y visualización de sistemas y aplicaciones paralelos de gran tamaño.

TAU aborda los problemas de rendimiento desde tres niveles: instrumentación, medición y análisis. De la misma manera, la exploración efectiva de rendimiento requiere seleccionar prudentemente un conjunto de métodos alternativos. Por ello, permite la combinación de experimentos de rendimiento significativos que facilitan la obtención de propiedades de rendimiento significativas. Para conseguir este objetivo, TAU ofrece la posibilidad de realizar el análisis del rendimiento de diferentes maneras, incluyendo la habilidad de crear un proceso de instrumentación robusto, modos de medición empleando trazado y generación de perfiles, análisis y gestión de datos de rendimiento [30].

Los datos obtenidos a través de los diferentes perfiles pueden visualizarse gráficamente en distintos tipos de gráficos estadísticos. Además, incluye un visualizador en tres dimensiones para facilitar la representación visual. La Ilustración 7 muestra como la capacidad de TAU para representar el tiempo consumido por las funciones de una aplicación en tres dimensiones.

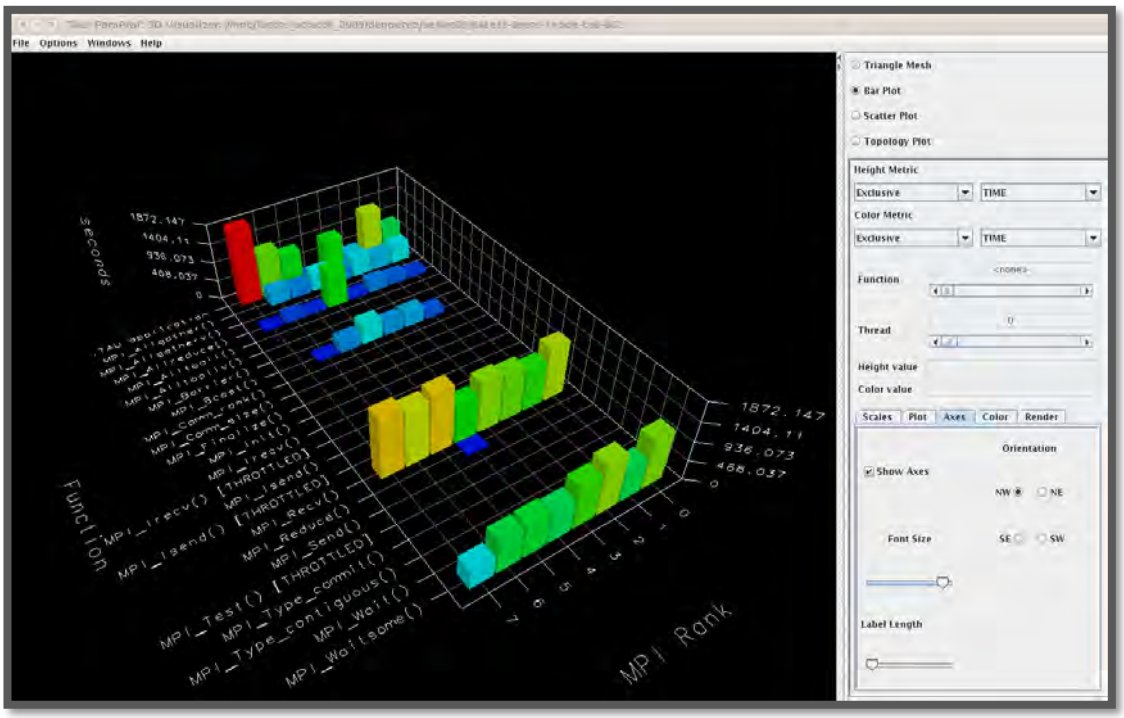



*Ilustración 7. Visualizador en tres dimensiones de TAU*

### 3.2.5 KOJAK

KOJAK supone un conjunto de herramientas diseñado para el análisis de rendimiento de aplicaciones paralelas basadas en MPI, *OpenMP* y aplicaciones híbridas.

Se compone de un módulo de generación de trazas y un módulo de instrumentación automática empleando TAU.

Su funcionamiento se basa en generar trazas de eventos para las aplicaciones en ejecución, para posteriormente determinar las posibles ineficiencias existentes en sus patrones de comportamiento.

El proceso de análisis convierte las trazas de eventos en representaciones en tres dimensiones del rendimiento que se almacena en perfiles. La primera dimensión describe el tipo de comportamiento que describe el rendimiento. La segunda dimensión presenta el comportamiento de las diferentes zonas del código fuente y la fase de la ejecución en la que se desarrollan. Por último, la tercera dimensión proporciona información acerca de la distribución de las pérdidas de rendimiento a través de los diferentes procesos o hilos. La organización jerárquica de las dimensiones permite investigar el comportamiento del rendimiento con niveles diferentes de granularidad [31].

Los perfiles generados por KOJAK se pueden visualizar empleando herramientas como CUBE, un componente de presentación gráfica de perfiles de ruta de llamadas [32]. La Ilustración 8 muestra la interfaz gráfica al visualizar un perfil de KOJAK en CUBE.

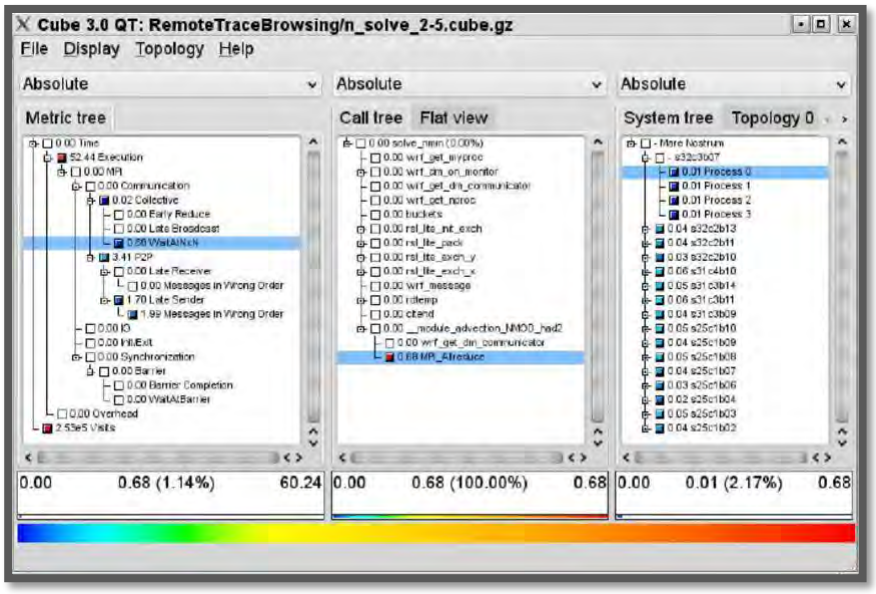



*Ilustración 8. Visualización en CUBE de un perfil de KOJAK*

### 3.2.6 Paradyn

*Paradyn* es una herramienta de medición de rendimiento para aplicaciones paralelas y distribuidas. Utiliza diversas tecnologías para escalar programas de larga duración y de tamaños grandes (abarcando miles de nodos). Además, automatiza la búsqueda de cuellos de botella en el rendimiento durante la ejecución. Es capaz de proveer información con alto nivel de precisión sobre el rendimiento a nivel de procedimiento y de sentencias.

Se fundamenta en la noción dinámica de la instrumentación y la medición. Sin necesidad de modificar los ficheros fuente de la aplicación analizada, se comienza la ejecución de la misma, y posteriormente la instrumentación del rendimiento es insertada en la aplicación durante su ejecución. La instrumentación se controla desde un módulo asesor que automáticamente dirige la misma y libera al desarrollador de tener que tomar decisiones sobre el control de datos dentro de la aplicación. Dicho módulo almacena información sobre la correlación entre cuellos de botella y la estructura de las aplicaciones, de manera que es capaz de determinar las causas de los cuellos de botella originados por partes específicas de una aplicación [33].

*Paradyn* puede ser configurado de manera sencilla para desarrollarse sobre diferentes sistemas operativos y *hardware*. Además, proporciona una interfaz de visualización para facilitar el análisis de los datos capturados. La Ilustración 9 muestra la interfaz gráfica de *Paradyn* para la visualización de datos.

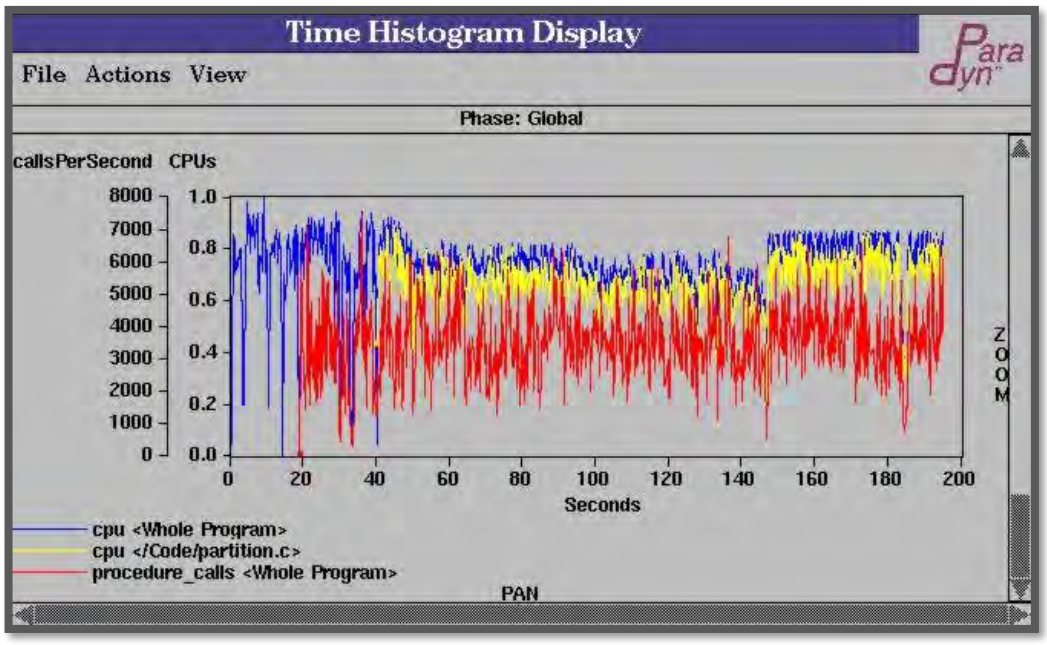



*Ilustración 9. Visualización gráfica de datos en forma de histograma en Paradyn*

### 3.2.7 Scalasca

*Scalasca* es un conjunto de herramientas de rendimiento que ha sido diseñado específicamente con el objetivo de analizar el comportamiento en ejecución de aplicaciones paralelas en sistemas de gran escala que cuenta con miles de procesadores. Soporta medición y análisis para aplicaciones que emplean MPI, *OpenMP* y estructuras híbridas. Antes de comenzar la recolección de datos de rendimiento, las aplicaciones deben ser correctamente instrumentadas [34].

Se fundamenta en un proceso de análisis de rendimiento incremental que integra resúmenes de tiempo de ejecución con estudios detallados del comportamiento concurrente de la aplicación a través de trazas de eventos.

Durante el análisis, *Scalasca* rastrea la existencia de patrones característicos que muestren estados de inactividad relacionados con las propiedades del rendimiento, clasifica las distintas instancias y cuantifica su importancia.

Este *software* proporciona la elección entre dos modos de análisis. El primer modo supone un análisis general, a modo de resumen, de la ruta de llamadas durante la ejecución. El segundo modo consiste en un estudio en profundidad del comportamiento siguiendo todos los eventos. En ambos modos, se cuenta con métricas de rendimiento para cada una de las llamadas producidas por procesos y *threads* que pueden ser examinados de forma interactiva en el reporte de análisis provisto.

Como alternativa a la búsqueda automática de patrones de comportamiento, las trazas pueden ser examinadas utilizando otros navegadores externos, como *Vampir,* aprovechando las funcionalidades de visualización gráfica.

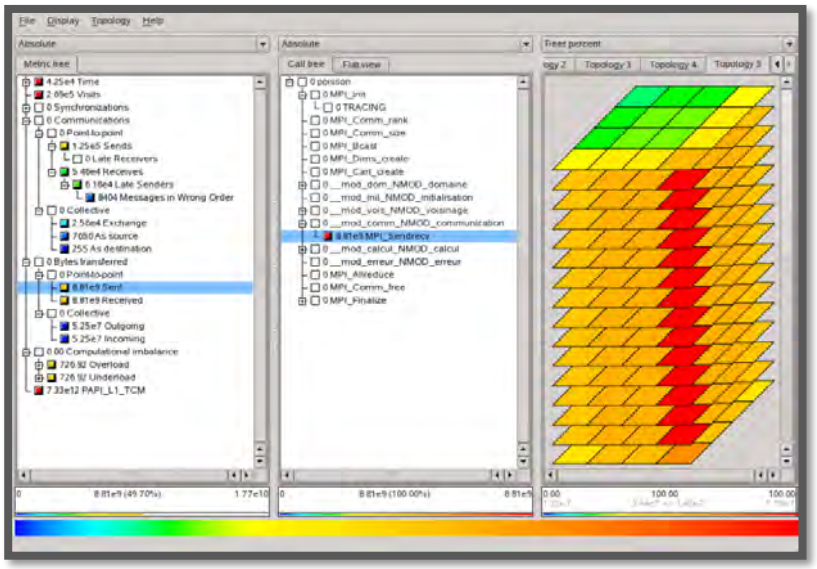


*Ilustración 10. Visualización gráfica de los datos con Scalasca*

### 3.2.8 PAPI

La herramienta PAPI especifica un interfaz de programación de aplicaciones estándar para acceder a los contadores de rendimiento hardware que se encuentran disponibles en la mayoría de los microprocesadores modernos. Estos contadores consisten en un conjunto de registros que capturan eventos que suceden en el microprocesador. Ante la monitorización de estos eventos se puede facilitar la correlación entre código fuente y objeto y la eficiencia de ese código respecto a la arquitectura en la que se ejecuta. Dicha correlación se puede utilizar para realizar ajustes manuales en el código, mejorar la optimización de la compilación y depurado, ajustar la monitorización y el modelo de rendimiento [35].

Además, la captura de estos eventos se puede utilizar para obtener información del comportamiento de las aplicaciones paralelas y en función del mismo, caracterizar sus patrones de actuación (aplicaciones intensivas en CPU, en accesos a memoria, en llamadas de rutinas de comunicación, etc.) y ajustar su uso de los recursos del sistema.

Proporciona dos interfaces subyacentes a los contadores hardware. La primera, consiste en una interfaz sencilla y de alto nivel que tiene como objetivo la adquisición de medidas sencillas. Simplemente proporciona la capacidad de iniciar, detener y leer eventos específicos, solo uno al mismo tiempo. La segunda interfaz es completamente programable, de bajo nivel, dirigida a la realización de medidas más sofisticadas. Esta interfaz trata los eventos hardware en grupos, los cuáles reflejan como son comúnmente usados los contadores, tomando medidas simultáneas y relacionándolas entre sí. Por ejemplo, la relación entre accesos a memoria y ciclos o entre fallos caché y *flops* puede indicar una mala gestión de memoria. Además, estos conjuntos de eventos permiten una implementación eficiente que se traduce en la realización de unas mediciones precisas y detalladas.

PAPI también facilita portabilidad para las diferentes plataformas en las que se pueda utilizar. Utiliza las mismas rutinas con las mismas listas de argumentos para acceder y controlar los contadores hardware de cada arquitectura.

Por defecto, cuenta con una serie conjunto de eventos predefinidos y estandarizados que no producen que el código fuente de la aplicación a monitorizar sea modificado. Si se desea acceder a eventos específicos la API de bajo nivel cuenta con todos los eventos y modos de captura disponibles en la arquitectura.

Por otra parte, si un evento no está disponible en una determinada arquitectura, PAPI indica un error informando del hecho, lo que permite ahorrar significativamente el esfuerzo de portar código a otras plataformas.

La Ilustración 11 presenta esquemáticamente la arquitectura de PAPI.

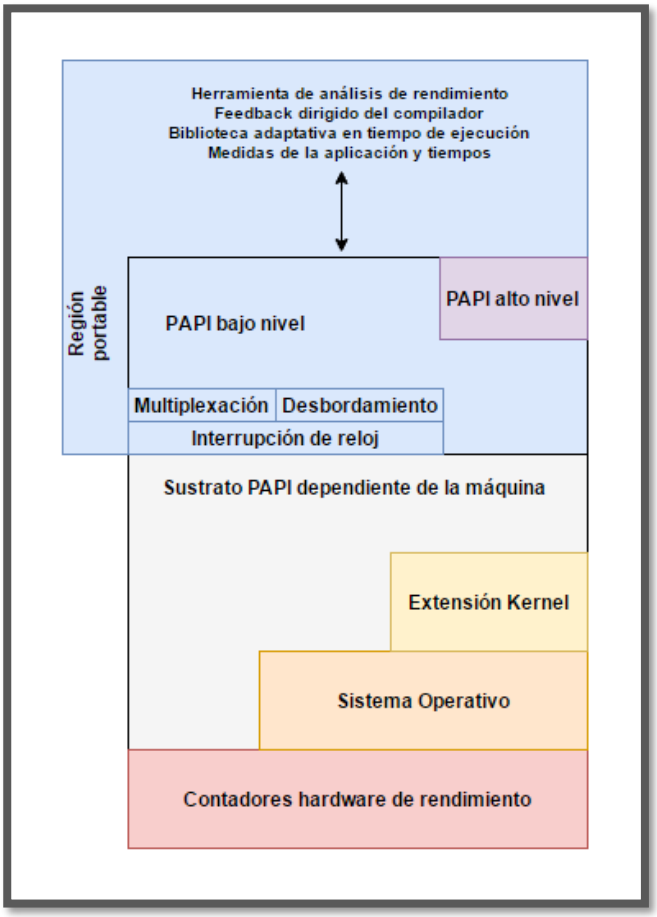



*Ilustración 11. La arquitectura de PAPI*

Su implementación se puede dividir en dos capas de software. La primera, se compone de la API y de las funciones independientes de la máquina. La segunda capa, define y exporta una interfaz independiente de la máquina a las funciones dependientes de la máquina y de las estructuras de datos. Estas funciones acceden al sistema operativo, a una extensión del *kernel* o bien a funciones de ensamblado para acceder directamente a los registros de los procesadores. PAPI intenta acceder al más eficiente de los tres, dependiendo de la disponibilidad de cada uno.

El desarrollo con PAPI proporciona ventajas en el desarrollo como la capacidad de medir numerosos contadores *hardware* con una interfaz común, la instrumentación de la aplicación una sola vez con cambios sencillos para la

migración a otras arquitecturas y la posibilidad, debido a sus características avanzadas, de ser utilizado en conjunción con otras herramientas de más alto nivel como *Vampir* o TAU.

Por todas las características que presenta, por encontrarse perfectamente integrado con FlexMPI y porque el interés del desarrollo es el de realizar una monitorización en tiempo de ejecución sin generar trazas de datos y sin utilizar aplicaciones o interfaces gráficas de terceros, sino que simplemente se desea utilizar una interfaz de bajo nivel para recolectar dinámicamente las medidas de rendimiento de las aplicaciones, se establece PAPI como la plataforma de monitorización utilizada para la realización del presente proyecto.

## 3.3 Benchmarks considerados

En el presente apartado se exponen los *benchmarks* considerados para el desarrollo del proyecto, exhibiendo sus principales características y diferencias y aportando un breve análisis de su comportamiento. Cabe destacar, que los *benchmarks* han sido ligeramente modificados para integrar las funcionalidades que proporciona la interfaz de alto nivel de FlexMPI y conseguir que sus operaciones se comportaran de una forma deseada.

### 3.3.1 Benchmark del método de gradiente conjugado

El método del gradiente conjugado es un algoritmo matemático que tiene el objetivo de resolver numéricamente sistemas de ecuaciones lineales, en los que las matrices que los componen son simétricas y definidas positivas.

El método del gradiente conjugado (CG) se implementa como un algoritmo iterativo, que normalmente se aplica a sistemas de ecuaciones donde la mayoría de sus elementos poseen un valor nulo y son demasiado grandes como para ser manipulados por otras implementaciones directas. Este tipo de sistemas de ecuaciones a menudo se derivan de la resolución de ecuaciones diferenciales parciales o de problemas de optimización [36].

Cabe destacar que el tiempo de ejecución de un *benchmark* de gradiente conjugado es directamente proporcional al tamaño de la matriz del sistema de ecuaciones a resolver e inversamente proporcional al número de procesos en los que se puede dividir la ejecución de la aplicación.

### 3.3.2 Benchmark del método de Jacobi

El método de Jacobi es un método iterativo, utilizado para la resolución de sistemas de ecuaciones lineales de matrices diagonales dominantes.

Se fundamenta en la construcción de una sucesión convergente que se define de manera iterativa. El límite de la sucesión construida, que precisamente es el valor al que converge, supone la solución del sistema. Se suele aplicar con matrices cuyos valores no son nulos en su mayoría.

Cabe destacar que la implementación del método de Jacobi es muy beneficiosa en *benchmarks* que integran mecanismos de paralelismo, pues al dividir la aplicación en un número determinado de procesos, el tiempo de ejecución decrece un valor prácticamente constante y proporcional al número de procesos utilizado. Además, puesto que el patrón de comportamiento de la implementación es regular, el tiempo de ejecución de cada iteración es en esencia constante.

Para conseguir distintas características y rendimientos, se puede variar el tamaño de las matrices de entrada, el número de iteraciones límite o el tiempo empleado en rutinas de comunicación, de manera que se obtienen *benchmarks* que sean más intensivos en la realización de operaciones en CPU, en los accesos a memoria o en llamadas a rutinas de bibliotecas.

La Ilustración 12 y la Ilustración 13 presentan el comportamiento del *benchmark* de Jacobi original frente al incremento del número de procesos y al incremento de los datos de entrada. Las pruebas se han realizado en un nodo del *cluster* Tucán que cuenta con dos procesadores de cuatro núcleos cada uno, y un tamaño de memoria principal de ocho gigabytes.

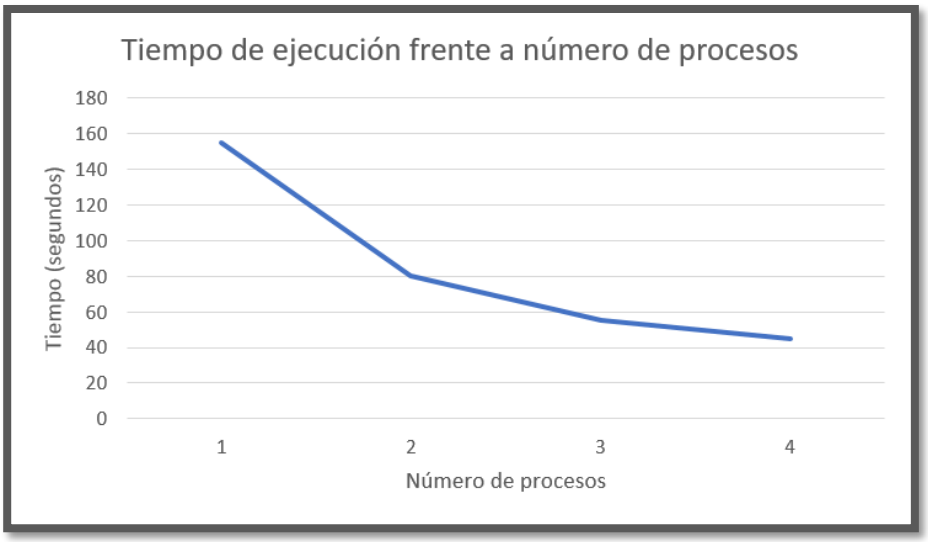



*Ilustración 12. Comportamiento del benchmark de Jacobi frente al incremento del número de procesos*

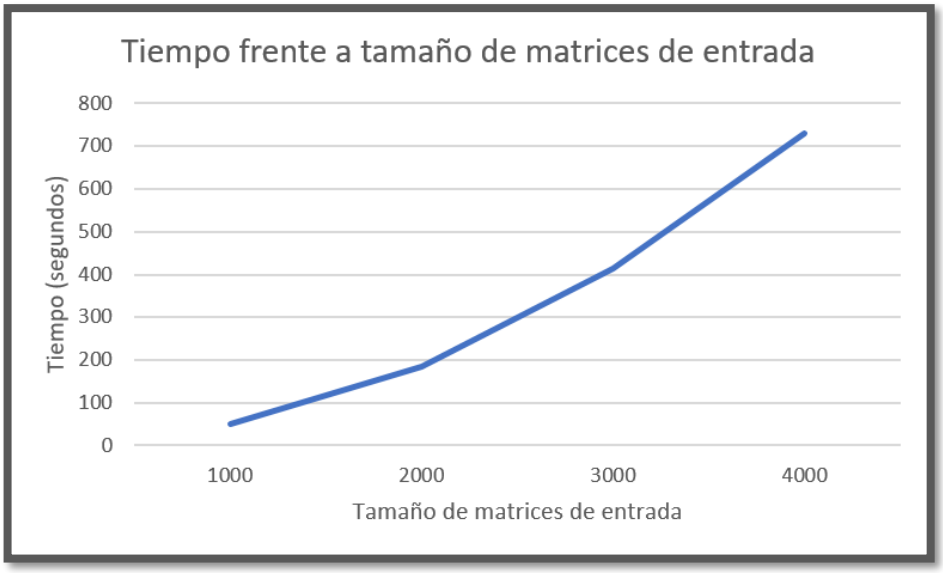



*Ilustración 13. Comportamiento del benchmark de Jacobi frente al incremento de la matriz de entrada*

Ante las características señaladas, la buena respuesta de paralelismo y la facilidad en la modificación del comportamiento para obtener *benchmarks* cuyas operaciones sean más significativas en procesamiento, memoria o comunicación, se establece Jacobi como el *benchmark* a utilizar en el desarrollo del presente proyecto para construir las aplicaciones paralelas del sistema.

### 3.3.3 Modificación del benchmark del método de Jacobi

Con el objetivo de disponer de aplicaciones paralelas que posean diferentes perfiles de comportamiento se han realizado diferentes modificaciones en el *benchmark* del método de Jacobi.

En primer lugar, las aplicaciones que tienen un comportamiento de realización de operaciones de procesamiento se construyen con la implementación del *benchmark* original con un tamaño de datos de entrada de cuatro kilobytes.

Para las aplicaciones con un comportamiento intensivo de acceso a memoria se aumenta el tamaño de los datos de entrada a dos megabytes.

Por último, para disponer de una aplicación intensiva en rutinas de comunicación se modifica la implementación del *benchmark* para que realice numerosas veces la operación nativa de MPI designada como *MPI_Allgatherv*. Esta operación permite comunicar datos entre todos los procesos basados en MPI de la aplicación a través del intercambio de mensajes. Este comportamiento permitirá que la aplicación consuma mucho tiempo en rutinas de comunicación en comparación con el tiempo dedicado al procesamiento. Las características de

paso de mensajes para llevar a cabo la comunicación entre procesos derivan en el hecho de que, cuanto mayor sea el número de procesos que forman la aplicación mayor será el tiempo que se invertirá en las rutinas de comunicación. Por tanto, las aplicaciones de comunicación no son escalables, a diferencia de las aplicaciones intensivas en procesamiento y en acceso a memoria. La Ilustración 14 muestra como la aplicación de comunicación aumenta su tiempo de ejecución al aumentar el número de procesos paralelos que la componen.

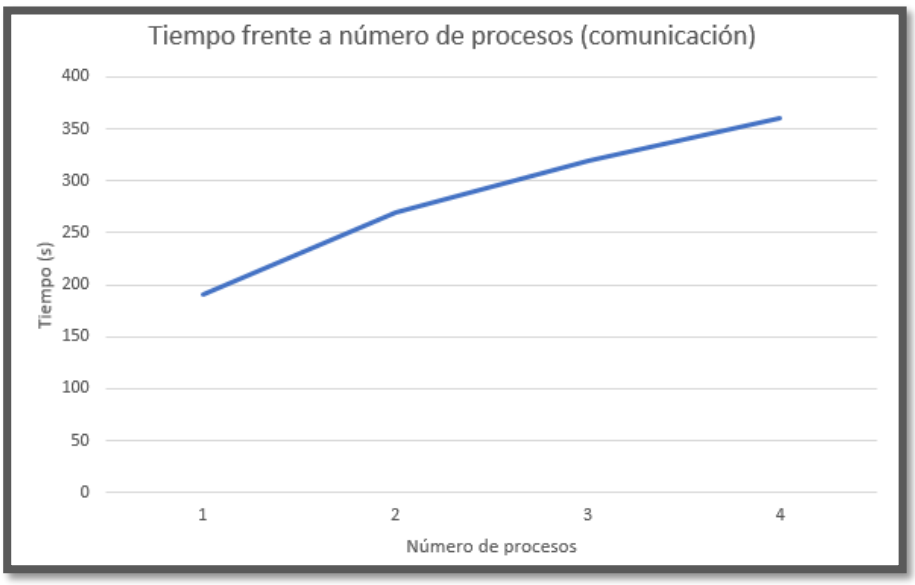



*Ilustración 14. Comportamiento del benchmark modificado de comunicación frente a número de procesos*

## 3.4 Entorno de desarrollo

En el presente apartado se presentan las diferentes herramientas que han contribuido en el desarrollo del proyecto. En primer lugar, se presentan los distintos sistemas operativos involucrados. En segundo lugar, se describen los lenguajes de programación utilizados. Posteriormente, exhiben las diferentes librerías externas que han participado en la realización del sistema. Por último, se exponen las distintas herramientas *software* que han servido de apoyo para la ejecución de las distintas tareas.

### 3.4.1 Sistemas operativos

A continuación, se presentan los sistemas operativos utilizados durante el desarrollo del presente proyecto y sus características más significativas.

- **Ubuntu 16.04 LTS.** En el entorno de desarrollo local se ha decido utilizar esta versión del sistema operativo *Ubuntu*. Las principales justificaciones que apoyan esta elección son la experiencia previa en el desarrollo con este sistema, la estabilidad que presenta esta versión (cuenta con la garantía de soporte a largo plazo, lo que significa que cuenta como mínimo con cinco años de actualizaciones y mantenimiento sin coste) y una comunidad de desarrollo muy extendida, lo que se traduce en la existencia de una mayor documentación e información ante posibles problemas que se pueden presentar. Consiste en un sistema operativo basado en GNU/*Linux* distribuido como *software* libre, incluyendo su propio entorno de escritorio que recibe el nombre de *Unity*. Las estadísticas indican que este sistema operativo es el más popular dentro de los sistemas *open source* para el desarrollo y despliegue de aplicaciones. Además, se refleja que *Ubuntu* es el sistema más utilizado para desarrollar dentro de las diferentes distribuciones *Linux*. La Ilustración 15 muestra los porcentajes de uso de las diferentes distribuciones de *Linux* para el desarrollo [38].

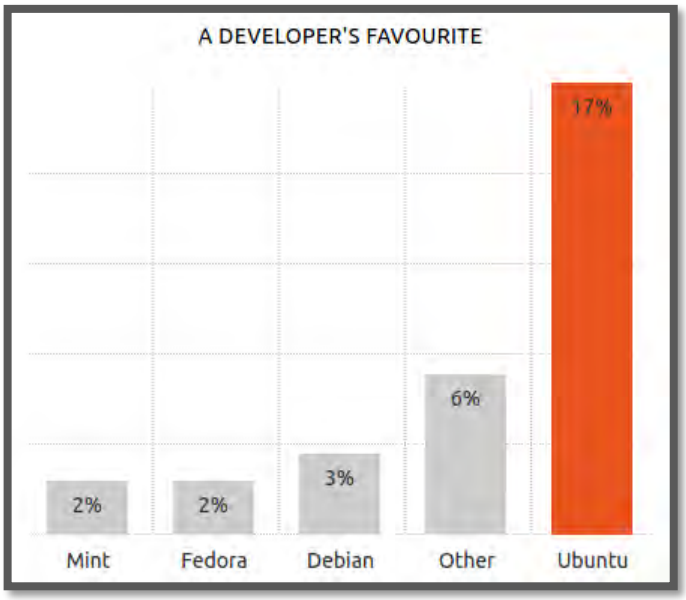



*Ilustración 15. Utilización de diferentes distribuciones Linux para el desarrollo*

- **Ubuntu Server 14.04.2 LTS.** Este sistema operativo es el que incorporan todos los nodos de computación en el sistema *cluster Tucán*. Es un sistema operativo basado en *Debian* distribuido también bajo la licencia GPL de *software* libre. No se podían presentar otras opciones de elección de sistema operativo para utilizar en el *cluster* puesto que es una tarea asignada al administrador del sistema.

### 3.4.2 Bibliotecas externas

A continuación, se presentan las bibliotecas de terceros que se utilizan a lo largo del desarrollo del proyecto.

En primer lugar, para hacer frente al desarrollo de las aplicaciones paralelas que se utilizan en el sistema se emplea MPICH en su versión 3.2, que como se ha indicado en el apartado 3.1.2 Versiones de MPI, consiste en una implementación gratuita del estándar MPI. Su cuidado interés por la portabilidad, su robustez y su mantenimiento lo convierten en la opción idónea para el desarrollo paralela del sistema a construir.

En cuanto a la monitorización del rendimiento de cada una de las aplicaciones paralelas existentes en el sistema se establece PAPI, en su versión 5.3.2.0 como la opción predilecta, cuyas características son comentadas en profundidad en la sección 3.2.8 PAPI. Es la opción que permite acceder a todos los contadores *hardware* disponibles en el sistema a través de dos tipos de interfaces, una a más alto nivel para implementaciones sencillas, y otra a bajo nivel que incluye mayores funcionalidades.

### 3.4.3 Lenguajes de programación

Para efectuar la elección del lenguaje de programación para desarrollar el presente proyecto se tuvo en cuenta los objetivos principales que debía alcanzar el sistema completo, asegurando que se cubría cada uno de ellos y que la funcionalidad buscada se mantuviese inalterada.

Considerando que las bibliotecas de MPI y PAPI con las que se trabajan están desarrolladas en *C* y buscando un buen rendimiento en la realización de las diferentes tareas paralelas que se necesitan para cumplir con los diferentes objetivos que se debían llevar a cabo, se estableció este lenguaje como el predominante para el desarrollo del proyecto.

Se trata de un lenguaje de propósito general considerado como uno de los más eficientes, puesto que combina las ventajas de los lenguajes de alto y bajo nivel reduciendo sus inconvenientes. Ha sido diseñado con el propósito de ser altamente portable y flexible.

Aunque su utilización puede resultar dificultosa por no ofrecer una abstracción a más alto nivel, debido a las primitivas de bajo nivel con las que trabaja, la experiencia previa adquirida con anteriores desarrollos, la consolida como una buena elección para la construcción del proyecto.

### 3.4.4 Software de desarrollo

En esta sección se describen todas las herramientas que han asistido en el desarrollo del *software* del proyecto.

- **GNU Nano.** Debido a las condiciones remotas del desarrollo y evaluación en el sistema *cluster Tucán*, se ha utilizado esta herramienta para realizar pequeñas modificaciones en los ficheros que conforman el proyecto. Se trata de un editor de texto para sistemas *Unix* basado en *Curses* (una biblioteca para control de terminales *Unix*, que hace posible la construcción de una interfaz de usuario para aplicaciones que se ejecutan en las mismas).

- **WinSCP.** Consiste en una aplicación cliente SFTP que dispone de una interfaz de usuario gráfica. Su función principal es la de facilitar la transferencia segura de archivos entre dos sistemas informáticos, uno local y otro remoto. De esta manera, se pueden modificar ficheros del desarrollo de manera local, reflejando los cambios automáticamente en el sistema *cluster Tucán* remoto.

- **Sublime Text.** Es un editor de texto y editor de código fuente desarrollado en *C++* y *Python*. Posee una interfaz de usuario gráfica apropiada para este proyecto y funcionalidades que facilitan el desarrollo de código en *C* y otros lenguajes. Ha sido utilizado para el desarrollo de código fuente local, apoyándose en *WinSCP* para sincronizar los ficheros modificados remotamente.

- **GCC.** Consiste en un conjunto de compiladores creados por el proyecto GNU. Son los compiladores estándares para los sistemas operativos derivados de Unix. Soporta una variedad de plataformas con el fin de promover el uso de un compilador global, capaz de atraer a múltiples equipos de desarrollo. Es el compilador utilizado a lo largo del proyecto para código *C*, en su versión 4.8.5, junto con la biblioteca del sistema *pthread* que define un conjunto de tipos definidos por el usuario, funciones y constantes.

## 3.5 Marco regulador

La presente sección recoge la combinación de los diferentes estatutos legales, reglas judiciales y la aplicación real que conviven con el desarrollo del presente proyecto

### 3.5.1 Legislación aplicable

Desde el punto de vista del entorno laboral, es necesario indicar que la utilización de FlexMPI está dirigida a miembros del ámbito profesional y que, por tanto, los usuarios deben poseer el suficiente conocimiento de programación en entornos de computación de alto rendimiento para comprender en qué medio se deben aplicar sus funcionalidades y la manera en la que se debe proceder para realizar el despliegue de la ejecución, evitando así los riesgos que se puedan derivar de su utilización.

Es necesario destacar que FlexMPI y las funcionalidades añadidas almacenan datos de monitorización de rendimiento que no suponen información privada, sensible o confidencial. Sin embargo, las aplicaciones paralelas que deben ser ejecutadas en el sistema podrían trabajar con datos personales, cuyo acceso está limitado a la propia aplicación. Por tanto, en España dichas aplicaciones deberán cumplir con las restricciones de la ley orgánica 15/1999 de protección de datos de carácter personal (LOPD), aprobada por el real decreto 1720/2007, cuyo objetivo es el de asegurar, en lo que concierne al tratamiento de los datos personales, las libertades públicas y derechos de las personas físicas, como son el poder de control de una persona sobre sus datos personales y sobre su uso y destino.

### 3.5.2 Estándares técnicos y propiedad intelectual

En primer lugar, es necesario analizar las licencias de las bibliotecas externas utilizadas por FlexMPI, que es la herramienta sobre la que añade funcionalidades el presente proyecto.

La implementación de MPI utilizada, MPICH se distribuye bajo una licencia BSD [42], otorgada principalmente para sistemas de tipo *Berkeley Software Distribution*, un tipo de sistema tipo *Unix*. Se trata de una licencia de software libre permisiva, muy flexible respecto a la distribución, en el que el *software*

puede ser redistribuido como *software* libre o propietario, siendo libre la licencia original del organismo autor.

Por otra parte, la institución autora de la biblioteca, con *Copyright* 2017 de La Universidad de Tennessee, PAPI, indica que permite su redistribución y uso, con o sin modificación siempre que se cumplan las siguientes condiciones [43]:

- Las redistribuciones del código fuente deben conservar el aviso de *Copyright* anterior esta lista de condiciones y la siguiente exención de responsabilidad.
- Las redistribuciones en formato binario deben conservar el aviso de *Copyright* anterior esta lista de condiciones y la siguiente exención de responsabilidad.
- Ni el nombre de los titulares de los derechos de autor ni los nombres de sus contribuyentes pueden ser utilizados para promover productos derivados de este *software* sin permiso previo y expreso por escrito.

Por tanto, FlexMPI y las funcionalidades añadidas se distribuirán con una licencia GPL, que supone una licencia muy popular en el ámbito del *software* libre y código abierto, sin entrar en conflicto con las licencias de las bibliotecas externas utilizadas.

## 3.6 Entorno socio-económico

En esta sección se describe los aspectos económicos y sociales vinculados a la elaboración del presente proyecto fin de grado. En primer lugar, se describirá el presupuesto asociado al desarrollo del proyecto y posteriormente se manifiesta el impacto social y económico esperado de la aplicación del resultado del proyecto.

### 3.6.1 Presupuesto

El desglose de los costes asociados al proyecto se divide en costes directos e indirectos.

- **Costes directos**

Suponen los costes asociados a contratación de personal y los costes derivados de los equipos y herramientas necesarios para desarrollar el proyecto.

Puesto que durante el desarrollo del proyecto se han asumido diversos roles dentro del mismo, los costes de personal se desglosan como muestra la Tabla 1. Se incluye el coste de la Seguridad Social referente a las horas comunes, ya que no se consideran horas extra.

| Rol | Tiempo dedicado (h) | Coste (€/h) | Porcentaje Seguridad social (Horas comunes) | Coste del Rol (€) |
|---|---|---|---|---|
| **Jefe de proyecto** | 81 | 38.50 | 23.60% | 3854.47 |
| **Analista** | 63 | 27.20 | 23.60% | 2118.01 |
| **Diseñador** | 54 | 34.00 | 23.60% | 2269.30 |
| **Programador** | 90 | 22.30 | 23.60% | 2480.66 |
| **Gestor de pruebas** | 72 | 22.10 | 23.60% | 1966.73 |
| **Total (€)** | 360 | - | - | **12689.17** |

*Tabla 1. Costes asociados al personal de trabajo*

Es necesario destacar que el despliegue del sistema y sus pruebas de rendimiento se realizaron en el *cluster Tucán* de la universidad Carlos III de Madrid, el cual se pudo utilizar de manera gratuita. Sin embargo, para establecer una mejor aproximación de los costes asociados, se incluyen los equipos utilizados en el desglose del presupuesto. La Tabla 2 y la Tabla 3 muestran los costes asociados al equipo y a las herramientas de trabajo respectivamente.

| Elemento | Coste unitario (€) | Unidades | Amortización por mes (€) | Coste total en 5 meses (€) |
|---|---|---|---|---|
| Nodo compute-1-2 | 5000.00 | 1 | 250.00 | 1250.00 |
| Nodo compute-1-7 | 5000.00 | 1 | 175.00 | 875.00 |
| Nodo compute-7-2 | 6000.00 | 1 | 125.25 | 626.25 |
| Ordenador personal | 1200.00 | 1 | 75.33 | 376.65 |
| **Total (€)** | - | - | - | **3126.90** |

*Tabla 2. Costes asociados al equipo*

| Elemento | Coste unitario (€) | Unidades | Coste total (€) |
|---|---|---|---|
| Hoja DinA4 | 0.02 | 100 | 2.00 |
| Bolígrafo | 0.05 | 10 | 0.50 |
| Archivador | 5.00 | 6 | 30.00 |
| **Total (€)** | - | - | **32.50** |

*Tabla 3. Costes asociados a las herramientas de trabajo*

- **Costes indirectos**

Suponen los costes relacionados indirectamente con el proyecto y que soportan su viabilidad.

Es necesario destacar que los equipos del *cluster* necesitan estar alojados en un centro de procesamiento de datos (CPD) así como otros costes, como electricidad y mantenimiento de los equipos. Por otra parte, es necesario añadir un coste de riesgo frente a imprevistos y problemas que se puedan originar durante el desarrollo del proyecto. La Tabla 4 presenta los costes indirectos asociados al presente proyecto.

| Descripción | Coste por mes (€) | Amortización por mes (€) | Coste Total en 5 meses (€) |
|---|---|---|---|
| Coste alquiler sala cluster | 1100.00 | 110.00 | 550.00 |
| Coste de energía y mantenimiento del CPD | 600.35 | 60.35 | 301.75 |
| Coste internet y teléfono | 60.33 | 5.33 | 26.65 |
| Coste de riesgo (10%) | - | - | 1672.70 |
| **Total** | | | **2551.10** |

*Tabla 4. Costes indirectos asociados al desarrollo*

Es necesario añadir el beneficio que se desea con el desarrollo del proyecto. Además, puesto que el presente proyecto se realiza en España el presupuesto está sujeto al impuesto de valor agregado o IVA. La Tabla 5 presenta de manera resumida el desglose del coste necesario para efectuar el presente proyecto.

| | |
|---|---|
| Coste de personal (€) | **12689.17** |
| Coste de equipos (€) | **3126.90** |
| Coste de herramientas (€) | **32.50** |
| Costes indirectos (€) | **2551.10** |
| Beneficio (€) | **5000.00** |
| Total, sin IVA (€) | 23719.70 |
| **Total, con IVA (21%) (€)** | **28700.84** |

*Tabla 5. Resumen del coste total asociado al proyecto*

### 3.6.2 Impacto socioeconómico

El presente apartado describe el impacto socioeconómico que se espera tras la aplicación del proyecto desarrollado.

Desde el punto de vista del impacto sobre el entorno social generado por FlexMPI junto con las funcionalidades añadidas, cabe destacar los beneficios aportados que contribuyen a un mejor despliegue de sistemas paralelos basados en memoria distribuida. La herramienta proporciona la capacidad de contar con una infraestructura de control central capaz de coordinar las aplicaciones que se desean ejecutar en el sistema, monitorizando sus métricas de rendimiento y permitiendo planificar su ejecución en función de los objetivos definidos por el usuario. Esta propuesta está destinada a su aplicación en centros de supercomputación en dónde se ejecutan un gran número de aplicaciones paralelas.

Teniendo en cuenta los aspectos económicos, es muy posible que en el futuro las contribuciones aportadas por el presente proyecto, junto a las funcionalidades originales de FlexMPI, puedan ser utilizadas en otros proyectos de investigación que cuenten con financiación económica y proyectos comerciales.

Sin embargo, como se refleja en el apartado 3.5 Marco regulador, la distribución de la biblioteca junto con las funcionalidades añadidas, se basará en una licencia GPL, que garantiza a los usuarios finales la libertad de utilizar, analizar, compartir y modificar el *software* desarrollado, protegiendo el hecho de que otros proyectos, aunque deriven en un beneficio económico, restrinjan esas libertades a los usuarios.

En resumen, el impacto socioeconómico directo sería nulo (debido a que supone un software libre) aunque indirectamente, mediante proyectos de investigación y colaboración con empresas se podría conseguir financiación. Dado que el fin último del proyecto es conseguir mejora en la eficiencia en el uso de grandes infraestructuras de computación paralela, en las que el consumo de energía puede ser muy elevado (del orden de megavatios), este aprovechamiento conlleva, de forma indirecta, un menor consumo de energía, con el consecuente impacto positivo medioambiental.

# Capítulo 4

# Propuesta

El presente capítulo tiene el propósito de describir el proyecto realizado en este trabajo fin de grado. En primer lugar, se presentan los objetivos perseguidos, la metodología y las fases de desarrollo. En segundo lugar, se especifican los requisitos exigidos para el producto *software* a realizar. Posteriormente, se describe el diseño del sistema desarrollado con la intención de proporcionar una abstracción de alto nivel del mismo. Por último, se muestra la planificación seguida en el proyecto.

## 4.1 Objetivos, metodología de trabajo y fases de desarrollo

En este apartado se exponen los objetivos perseguidos por el proyecto a realizar, así como la metodología seguida y las distintas fases por las que se condujo el desarrollo.

Es necesario destacar que un marco de trabajo de desarrollo común cuenta con dos roles predefinidos: un cliente, que demanda la creación de un sistema conforme a sus necesidades e intereses y un equipo de desarrollo, que se encarga de estructurar, planificar y controlar el proceso de desarrollo del sistema. En esta ocasión, el tutor del trabajo fin de grado adoptará el rol de cliente, a la vez que participará en el diseño de la propuesta, mientras que el alumno se encargará de asumir los diferentes cometidos que conciernen al equipo de desarrollo.

La metodología de desarrollo seguida se ha basado en el modelo-V de desarrollo, que es considerado una extensión del modelo en cascada, en el cual se ordena rigurosamente cada etapa del desarrollo, de tal manera que el inicio de cada fase debe esperar a la finalización de la anterior. En el modelo-V, en lugar de progresar de una manera secuencial rigurosa, los pasos de verificación y

mantenimiento se superponen en forma de *V*, de manera que, si una de las fases de validación no se desarrolla correctamente, se deben reajustar las fases de desarrollo siguiendo la forma que da nombre al modelo [39]. La Ilustración 16 presenta de manera esquemática el ciclo de vida de esta metodología de desarrollo.

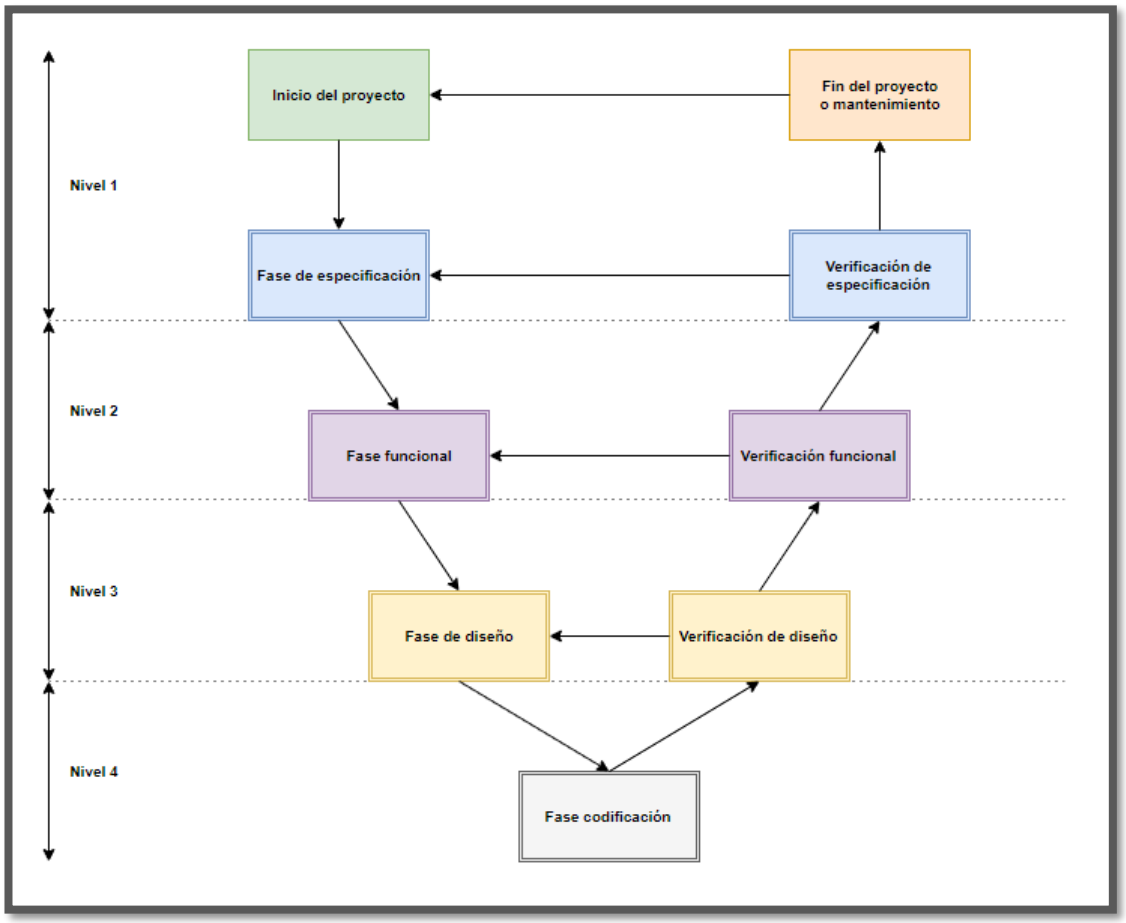



*Ilustración 16. Ciclo de vida en el modelo-V*

Para cada fase de desarrollo existe una fase correspondiente de verificación, de manera que se obedece el principio de ingeniería software de que para cada fase de desarrollo debe coexistir un resultado verificable. La distancia en el eje horizontal representa proporcionalmente la distancia en el tiempo que existe entre una fase de desarrollo y su homóloga de verificación, mientras que la distancia en el eje vertical representa el nivel de abstracción alcanzado en el proyecto. El primer nivel se orienta al cliente, ya que constituye las fases de especificación y análisis de requisitos. El segundo nivel define las características funcionales del sistema propuesto, considerando al sistema como una caja negra. El tercer nivel se fundamenta en el diseño de los componentes *software* y

*hardware* del sistema final. Por último, el cuarto nivel se corresponde con la implementación de todos los elementos y módulos del sistema.

Aunque esta metodología de desarrollo puede resultar muy estricta y rígida, pues para cada cambio se deben revisar todas las fases de desarrollo inferiores a la fase modificada, resulta una metodología sencilla de aplicar, en la que los requisitos son claros antes de comenzar el desarrollo y cada fase se completa antes de iniciar la siguiente, asegurando la corrección deseada en el sistema completo.

Durante el desarrollo, se concertaban diversas reuniones con el cliente, para asegurar que el sistema contaba con las funcionalidades esperadas, y que, además, marcaban el ritmo en el que se iniciaban y finalizaban las distintas etapas descritas.

En primer lugar, se acordó una reunión introductoria, llevada a cabo la última semana de noviembre de 2016, en la que se estableció el estado de la cuestión abordada por el proyecto y los principales objetivos perseguidos por el mismo.

Los objetivos del proyecto son:

- Análisis y despliegue de FlexMPI.
- Añadir nuevas funcionalidades a FlexMPI.
- Desarrollo de un conjunto de *benchmarks* representativo de aplicaciones paralelas e integración de los mismos con MPI.
- Desarrollo de una infraestructura para el análisis centralizado de las métricas monitorizadas de una aplicación paralela.
- Desarrollo de una infraestructura capaz de monitorizar el estado de la carga de trabajo del sistema y planificar los siguientes trabajos a ejecutar.
- Desarrollo de técnicas de mapeo entre procesos y núcleos de procesamiento.
- Desarrollo de una política de maleabilidad.
- Desarrollo de una política de sobresuscripción.
- Evaluación de la propuesta.

Posteriormente, a partir de la última semana de enero de 2017, se acordaron reuniones semanales alternadamente, que abordaban las fases de elicitación, especificación y análisis de requisitos, diseño de la arquitectura e implementación, junto con sus fases de verificación, reiniciando el ciclo de vida al introducir ajustes que corrigieran cada una de las etapas que no alcanzaban los requisitos esperados.

## 4.2 Análisis de requisitos

Con el objetivo de exponer los diferentes requisitos del sistema, se seguirán las recomendaciones proporcionadas por la IEEE para la especificación de requisitos *software* (SRS) [40].

Dichas recomendaciones fundamentan que los requisitos deben apuntar cuestiones básicas, tales como qué hace el sistema, cómo interactúa el sistema con usuarios u otros sistemas, qué disponibilidad y calidad de servicio debe ofrecer, cuáles son las consideraciones de portabilidad, mantenibilidad y seguridad y a qué estándares o límites del entorno se atiene. Por su parte, en la forma de presentación, la especificación debe ser correcta, conteniendo sólo requisitos que el sistema debe cumplir, no ambigua, completa, conteniendo todos los requisitos que el sistema necesita, consistente, sin presentar conflictos, verificable, asumiendo la existencia de un proceso capaz de verificar cada uno de los requisitos de la especificación, modificable, aplicando cambios que permiten mantener la estructura y el estilo y trazable, asumiendo un claro origen de los distintos requisitos y permitiendo su seguimiento en etapas futuras del desarrollo.

Además, para asegurar el principio de ingeniería de software que determina que se debe hacer una clara separación entre los diferentes niveles de descripción, se obtendrán en primer lugar los requisitos de usuario, aquellos exigidos por el cliente, de los que se derivan los requisitos software o del sistema, que nacen de la necesidad del cumplimiento de lo exigido por el cliente en un plano más técnico.

Los requisitos se presentarán en forma tabular. La Tabla 6 muestra la plantilla que seguirán los requisitos expuestos.

| | | | |
|---|---|---|---|
| **ID** | RU-XX, RF-XX o RNF-XX | | |
| **Título** | Nombre del requisito | | |
| **Descripción** | Descripción del requisito | | |
| **Relación** | Relación del presente requisito con requisitos de distinto tipo | | |
| **Prioridad** | Alta, media o baja | **Fuente** | Cliente o desarrollador |
| **Necesidad** | Esencial, deseable u opcional | **Claridad** | Alta, media o baja |
| **Estabilidad** | Alta, media o baja | **Verificabilidad** | Alta, media o baja |

*Tabla 6. Plantilla tabular para la presentación de requisitos*

Los atributos que definen un requisito son los siguientes:

- **ID:** código que identifica de forma unívoca a cada requisito (RU significa requisito de usuario, RF requisito software funcional y RNF requisito software no funcional y XX un número de dos dígitos).
- **Título:** nombre que describe de forma breve al requisito.
- **Descripción:** explicación y redacción del requisito.
- **Relación:** mide la relación con requisitos de usuario o con requisitos software. Contendrá el identificador de aquellos recursos con los que tiene relación el requisito.
- **Prioridad:** medida que determina la urgencia del requisito en el desarrollo. Puede ser alta, media o baja.
- **Fuente:** define el origen del requisito. Puede provenir del cliente o bien del propio equipo de desarrollo.
- **Necesidad:** medida del interés de los usuarios/clientes en que el sistema realice el requisito. Puede ser Esencial en su mayor grado, deseable en término medio u opcional en un grado bajo.
- **Claridad:** medida de la ambigüedad del requisito y su correcta formulación lingüística. Puede ser alta, media o baja.
- **Estabilidad:** medida sobre el valor de permanencia del requisito a lo largo de la vida del software. Puede ser alta (durante toda la vida del sistema), media o baja (hasta que se cumpla un determinado evento en el sistema, como la salida de una nueva versión o estándar).
- **Verificabilidad:** medida del nivel de comprobación que admite un requisito. Puede ser alta, media o baja.

### 4.2.1 Requisitos de usuario

Los requisitos de usuario suponen declaraciones en lenguaje natural, sin tecnicismos, de los servicios que exige el cliente y que se espera que el sistema provea. Especifican el comportamiento externo del sistema evitando describir características internas al diseño del sistema.

A continuación, se presentan los diferentes requisitos de usuario recogidos para el presente proyecto.

| | | | |
|---|---|---|---|
| **ID** | RU-01 | | |
| **Título** | El sistema tendrá aplicaciones paralelas intensivas en uso de cpu | | |
| **Descripción** | El sistema tendrá aplicaciones paralelas cuyo uso predominante de los recursos sean operaciones de procesamiento | | |
| **Relación** | RNF-01 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Media |

*Tabla 7. Requisito de usuario RU-01*

| | | | |
|---|---|---|---|
| **ID** | RU-02 | | |
| **Título** | El sistema tendrá aplicaciones paralelas intensivas en uso de memoria | | |
| **Descripción** | El sistema tendrá aplicaciones paralelas cuyo uso predominante de los recursos sean operaciones de acceso a memoria | | |
| **Relación** | RNF-02 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Media |

*Tabla 8. Requisito de usuario RU-02*

| | | | |
|---|---|---|---|
| **ID** | RU-03 | | |
| **Título** | El sistema tendrá aplicaciones paralelas intensivas en uso de rutinas de comunicación | | |
| **Descripción** | El sistema tendrá aplicaciones paralelas cuyo uso predominante de los recursos sean llamadas a rutinas de comunicación | | |
| **Relación** | RNF-03 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Media |

*Tabla 9. Requisito de usuario RU-03*

| | | | |
|---|---|---|---|
| **ID** | RU-04 | | |
| **Título** | El sistema tendrá una carga de trabajo estática | | |
| **Descripción** | El sistema tendrá una carga de trabajo estática formada por aplicaciones paralelas y el número mínimo de procesos de los que se componen | | |
| **Relación** | RF-01, RF-04, RF-05, RF-06, RF-12, RNF-04, RNF-05 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Baja |
| **Estabilidad** | Alta | **Verificabilidad** | Media |

*Tabla 10. Requisito de usuario RU-04*

| | | | |
|---|---|---|---|
| **ID** | RU-05 | | |
| **Título** | Las aplicaciones paralelas se dividirán en tantos procesos como indique la carga de trabajo | | |
| **Descripción** | Las aplicaciones paralelas se dividirán en tantos procesos como indique la carga de trabajo, con un mínimo de un proceso por aplicación y hasta un máximo de número de procesos dado por el número de núcleos del sistema | | |
| **Relación** | RF-01, RF-12, RF-14 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 11. Requisito de usuario RU-05*

| | | | |
|---|---|---|---|
| **ID** | RU-06 | | |
| **Título** | El sistema tendrá un controlador | | |
| **Descripción** | El sistema tendrá una infraestructura central de control | | |
| **Relación** | RF-01, RF-02, RF-03 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Media |

*Tabla 12. Requisito de usuario RU-06*

| | | | |
|---|---|---|---|
| **ID** | RU-07 | | |
| **Título** | El controlador podrá comunicarse remotamente con las aplicaciones paralelas | | |
| **Descripción** | El controlador tendrá la capacidad de comunicarse con las aplicaciones paralelas de forma remota en el sistema para poder gestionarlas | | |
| **Relación** | RF-15, RF-16, RF-18, RF-20, RF-21, RF-22 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 13. Requisito de usuario RU-07*

| | | | |
|---|---|---|---|
| **ID** | RU-08 | | |
| **Título** | El controlador monitorizará el número de núcleos de procesamiento libres en el sistema | | |
| **Descripción** | El controlador monitorizará el número de núcleos de procesamiento que no están ejecutando ningún proceso del sistema | | |
| **Relación** | RF-13 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 14. Requisito de usuario RU-08*

| | | | |
|---|---|---|---|
| **ID** | RU-09 | | |
| **Título** | El controlador podrá lanzar las aplicaciones paralelas remotamente | | |
| **Descripción** | La infraestructura de control podrá ejecutar las aplicaciones paralelas de la carga de trabajo del sistema remotamente | | |
| **Relación** | RF-14 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 15. Requisito de usuario RU-09*

| | | | |
|---|---|---|---|
| **ID** | RU-10 | | |
| **Título** | El controlador monitorizará el estado de las aplicaciones paralelas durante su ejecución | | |
| **Descripción** | La infraestructura de control debe monitorizar el estado de las aplicaciones paralelas durante su ejecución | | |
| **Relación** | RF-16, RF-17 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 16. Requisito de usuario RU-10*

| | | | |
|---|---|---|---|
| **ID** | RU-11 | | |
| **Título** | El controlador almacenará los resultados más recientes de monitorización de las aplicaciones paralelas | | |
| **Descripción** | La infraestructura de control debe almacenar los resultados más recientes monitorizados de las aplicaciones paralelas, hasta un máximo de diez resultados | | |
| **Relación** | RF-17 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 17. Requisito de usuario RU-11*

| | | | |
|---|---|---|---|
| **ID** | RU-12 | | |
| **Título** | El controlador monitorizará la finalización de una aplicación | | |
| **Descripción** | La infraestructura de control monitorizará el momento en el que la aplicación termina su ejecución | | |
| **Relación** | RF-18 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 18. Requisito de usuario RU-12*

| | | | |
|---|---|---|---|
| **ID** | RU-13 | | |
| **Título** | El controlador podrá planificar la ejecución de aplicaciones siguiendo un orden secuencial | | |
| **Descripción** | La infraestructura de control podrá planificar la ejecución de aplicaciones siguiendo el orden secuencial definido en la carga de trabajo estática | | |
| **Relación** | RF-04, RF-14 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 19. Requisito de usuario RU-13*

| | | | |
|---|---|---|---|
| **ID** | RU-14 | | |
| **Título** | El controlador podrá planificar la ejecución de aplicaciones fuera del orden secuencial | | |
| **Descripción** | La infraestructura de control podrá planificar la ejecución de aplicaciones fuera del orden secuencial definido en la carga de trabajo estática | | |
| **Relación** | RF-05, RF-14 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 20. Requisito de usuario RU-14*

| | | | |
|---|---|---|---|
| **ID** | RU-15 | | |
| **Título** | El controlador podrá incrementar el número de procesos asociados a una aplicación paralela durante su ejecución | | |
| **Descripción** | El controlador debe poder incrementar el número de procesos asociados a una aplicación paralela durante su ejecución, hasta un máximo de procesos dado por el número de núcleos del sistema | | |
| **Relación** | RF-06, RF-19, RF-20 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 21. Requisito de usuario RU-15*

| | | | |
|---|---|---|---|
| **ID** | RU-16 | | |
| **Título** | El controlador podrá aumentar el número de procesos de las aplicaciones en caso de que existan núcleos de procesamiento libres | | |
| **Descripción** | El controlador debe poder aumentar el número de procesos de las aplicaciones en caso de que existan núcleos de procesamiento que no ejecuten ningún otro proceso de aplicaciones paralelas del sistema | | |
| **Relación** | RF-06, RF-13, RF-19, RF-20 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 22. Requisito de usuario RU-16*

| | | | |
|---|---|---|---|
| **ID** | RU-17 | | |
| **Título** | El controlador mapeará los procesos de las aplicaciones paralelas en núcleos de procesamiento del sistema libres preferentemente | | |
| **Descripción** | El controlador mapeará los procesos de las aplicaciones paralelas en núcleos de procesamiento del sistema que no ejecuten otros procesos de aplicaciones paralelas preferentemente | | |
| **Relación** | RF-13, RF-14, RF-21 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Media |

*Tabla 23. Requisito de usuario RU-17*

| | | | |
|---|---|---|---|
| **ID** | RU-18 | | |
| **Título** | El controlador podrá mapear los procesos de las aplicaciones paralelas en un mismo núcleo de procesamiento del sistema, hasta un máximo de dos procesos por núcleo | | |
| **Descripción** | El controlador podrá mapear los procesos de las aplicaciones paralelas en un mismo núcleo de procesamiento del sistema, hasta un máximo de dos procesos por núcleo | | |
| **Relación** | RF-07, RF-13, RF-14 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 24. Requisito de usuario RU-18*

| | | | |
|---|---|---|---|
| **ID** | RU-19 | | |
| **Título** | El controlador permitirá seleccionar una política de planificación secuencial | | |
| **Descripción** | El controlador tendrá la capacidad de seleccionar la política de planificación que se utilizará para determinar el orden de ejecución de las aplicaciones del sistema | | |
| **Relación** | RF-02, RF-03 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 25. Requisito de usuario RU-19*

| | | | |
|---|---|---|---|
| **ID** | RU-20 | | |
| **Título** | El controlador permitirá seleccionar una política de planificación fuera de orden | | |
| **Descripción** | El controlador tendrá la capacidad de seleccionar una política de planificación fuera del orden establecido en la carga de trabajo del sistema | | |
| **Relación** | RF-02, RF-03 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 26. Requisito de usuario RU-20*

| | | | |
|---|---|---|---|
| **ID** | RU-21 | | |
| **Título** | El controlador permitirá seleccionar una política de planificación fuera de orden con maleabilidad | | |
| **Descripción** | El controlador tendrá la capacidad seleccionar una política de planificación fuera del orden establecido en la carga de trabajo del sistema, añadiendo maleabilidad de procesos | | |
| **Relación** | RF-02, RF-03 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 27. Requisito de usuario RU-21*

| | | | |
|---|---|---|---|
| **ID** | RU-22 | | |
| **Título** | El controlador permitirá seleccionar una política de planificación de sobresuscripción de procesos en los núcleos de procesamiento del sistema | | |
| **Descripción** | El controlador tendrá la capacidad de seleccionar una política de planificación de sobresuscripción de procesos en los núcleos de procesamiento del sistema | | |
| **Relación** | RF-02, RF-03 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 28. Requisito de usuario RU-22*

| | | | |
|---|---|---|---|
| **ID** | RU-23 | | |
| **Título** | El controlador permitirá suspender la planificación de aplicaciones paralelas | | |
| **Descripción** | El controlador tendrá la capacidad de pausar la planificación de aplicaciones paralelas del sistema que se encuentre en ejecución en ese momento | | |
| **Relación** | RF-02, RF-10 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 29. Requisito de usuario RU-23*

| **ID** | RU-24 | | |
|---|---|---|---|
| **Título** | El controlador permitirá reanudar la planificación de aplicaciones paralelas | | |
| **Descripción** | El controlador tendrá la capacidad de reanudar la planificación de aplicaciones paralelas del sistema que se encuentre en suspensión | | |
| **Relación** | RF-02, RF-11 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 30. Requisito de usuario RU-24*

| **ID** | RU-25 | | |
|---|---|---|---|
| **Título** | El controlador podrá forzar la terminación de las aplicaciones paralelas | | |
| **Descripción** | El controlador tendrá la capacidad de terminar la ejecución de las aplicaciones paralelas del sistema | | |
| **Relación** | RF-22, RF-23 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 31. Requisito de usuario RU-25*

| **ID** | RU-26 | | |
|---|---|---|---|
| **Título** | El controlador mostrará mensajes de error durante la ejecución | | |
| **Descripción** | El controlador mostrará mensajes de los errores producidos durante la ejecución del sistema | | |
| **Relación** | RF-24 | | |
| **Prioridad** | Media | **Fuente** | Cliente |
| **Necesidad** | Deseable | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 32. Requisito de usuario RU-26*

| **ID** | RU-27 | | |
|---|---|---|---|
| **Título** | El controlador mostrará mensajes de información de la inicialización del sistema | | |
| **Descripción** | El controlador mostrará mensajes de información de la inicialización del sistema indicando el estado de cada una de las tareas de dicho proceso | | |
| **Relación** | RF-01, RF-25 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 33. Requisito de usuario RU-27*

| | | | |
|---|---|---|---|
| **ID** | RU-28 | | |
| **Título** | El controlador mostrará mensajes de información de la posibilidad de introducir comandos | | |
| **Descripción** | El controlador mostrará mensajes de información de la posibilidad de introducir comandos por parte del usuario para aplicar instrucciones en la aplicación | | |
| **Relación** | RF-02, RF-26 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 34. Requisito de usuario RU-28*

| | | | |
|---|---|---|---|
| **ID** | RU-29 | | |
| **Título** | El controlador mostrará mensajes de información de la política de planificación seleccionada | | |
| **Descripción** | El controlador mostrará mensajes indicando la política de planificación seleccionada ante la selección de una política | | |
| **Relación** | RF-26 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 35. Requisito de usuario RU-29*

| | | | |
|---|---|---|---|
| **ID** | RU-30 | | |
| **Título** | El controlador mostrará mensajes de información del arranque del sistema | | |
| **Descripción** | El controlador mostrará mensajes de información indicando el comienzo de la planificación | | |
| **Relación** | RF-26 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 36. Requisito de usuario RU-30*

| | | | |
|---|---|---|---|
| **ID** | RU-31 | | |
| **Título** | El controlador mostrará mensajes de información de la suspensión del sistema | | |
| **Descripción** | El controlador mostrará mensajes indicando la suspensión del sistema cuando se produzca | | |
| **Relación** | RF-27 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 37. Requisito de usuario RU-31*

| | | | |
|---|---|---|---|
| **ID** | RU-32 | | |
| **Título** | El controlador mostrará mensajes de información de la reanudación del sistema | | |
| **Descripción** | El controlador mostrará mensajes indicando la reanudación del sistema cuando se produzca | | |
| **Relación** | RF-28 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 38. Requisito de usuario RU-32*

| | | | |
|---|---|---|---|
| **ID** | RU-33 | | |
| **Título** | El controlador mostrará mensajes de información del inicio de una aplicación paralela | | |
| **Descripción** | El controlador mostrará mensajes de información indicando el inicio de la ejecución de una aplicación paralela del sistema | | |
| **Relación** | RF-29 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 39. Requisito de usuario RU-33*

| | | | |
|---|---|---|---|
| **ID** | RU-34 | | |
| **Título** | El controlador mostrará mensajes de información de la finalización de una aplicación paralela | | |
| **Descripción** | El controlador mostrará mensajes de información indicando la finalización de una aplicación paralela del sistema | | |
| **Relación** | RF-30 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 40. Requisito de usuario RU-34*

| | | | |
|---|---|---|---|
| **ID** | RU-35 | | |
| **Título** | El controlador mostrará mensajes de información del incremento de procesos de una aplicación paralela | | |
| **Descripción** | El controlador mostrará mensajes de información indicando el incremento de procesos de una aplicación paralela del sistema cuando se aplique maleabilidad | | |
| **Relación** | RF-31 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 41. Requisito de usuario RU-35*

| | | | |
|---|---|---|---|
| **ID** | RU-36 | | |
| **Título** | El controlador mostrará mensajes de información del mapeo entre procesos y núcleos de procesamiento del sistema | | |
| **Descripción** | El controlador mostrará mensajes de información indicando la asignación de procesos a núcleos de procesamiento del sistema | | |
| **Relación** | RF-32 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 42. Requisito de usuario RU-36*

| | | | |
|---|---|---|---|
| **ID** | RU-37 | | |
| **Título** | El controlador mostrará mensajes de información de la finalización de la planificación del sistema completo | | |
| **Descripción** | El controlador mostrará mensajes de información indicando la finalización de la planificación del sistema completo tras haber sido completada | | |
| **Relación** | RF-33 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 43. Requisito de usuario RU-37*

### 4.2.1 Requisitos del sistema

Los requisitos del sistema toman en cuenta los requisitos de usuario para establecer con detalle los servicios y restricciones del sistema. Son descripciones más precisas de los requisitos de usuario. Son utilizados como punto de partida para el diseño del sistema. Se pueden dividir en dos categorías: los requisitos funcionales, que suponen atributos de funcionalidad del sistema y cuentan con un conjunto de entradas, cálculos y salidas, y los requisitos no funcionales, que suponen atributos de calidad que describen características del funcionamiento.

#### 4.2.1.1 Requisitos funcionales

A continuación, se presentan los requisitos del sistema funcionales recogidos para el presente proyecto.

| | | | |
|---|---|---|---|
| **ID** | RF-01 | | |
| **Título** | El controlador inicializará el sistema teniendo en cuenta la carga de trabajo | | |
| **Descripción** | El controlador percibirá la carga de trabajo e inicializará los recursos necesarios del sistema | | |
| **Relación** | RU-04, RU-05, RU-06, RU-27 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 44. Requisito funcional RF-01*

| | | | |
|---|---|---|---|
| **ID** | RF-02 | | |
| **Título** | El controlador obtendrá continuamente la introducción de datos por entrada estándar | | |
| **Descripción** | El controlador obtendrá continuamente la introducción de comandos por entrada estándar para poder aplicar acciones e instrucciones | | |
| **Relación** | RU-06, RU-19, RU-20, RU-21, RU-22, RU-23, RU-24, RU-28 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 45. Requisito funcional RF-02*

| | | | |
|---|---|---|---|
| **ID** | RF-03 | | |
| **Título** | El controlador obtendrá la política de planificación a aplicar por entrada estándar | | |
| **Descripción** | El controlador obtendrá la política de planificación a aplicar como una instrucción por entrada estándar | | |
| **Relación** | RU-06, RU-19, RU-20, RU-21, RU-22 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 46. Requisito funcional RF-03*

| | | | |
|---|---|---|---|
| **ID** | RF-04 | | |
| **Título** | El controlador podrá planificar secuencialmente la ejecución de las aplicaciones paralelas indicadas en la carga de trabajo | | |
| **Descripción** | El controlador tendrá la posibilidad de ejecutar las aplicaciones en el orden especificado en la carga de trabajo | | |
| **Relación** | RU-04, RU-13 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 47. Requisito funcional RF-04*

| | | | |
|---|---|---|---|
| **ID** | RF-05 | | |
| **Título** | El controlador podrá planificar fuera de orden la ejecución de las aplicaciones paralelas indicadas en la carga de trabajo | | |
| **Descripción** | El controlador tendrá la posibilidad de ejecutar las aplicaciones fuera del orden especificado en la carga de trabajo | | |
| **Relación** | RU-04, RU-14 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 48. Requisito funcional RF-05*

| | | | |
|---|---|---|---|
| **ID** | RF-06 | | |
| **Título** | El controlador podrá planificar fuera de orden con maleabilidad de procesos la ejecución de las aplicaciones paralelas indicadas en la carga de trabajo | | |
| **Descripción** | El controlador tendrá la posibilidad de ejecutar las aplicaciones fuera del orden especificado en la carga de trabajo y de incrementar su número de procesos en caso de tener núcleos de procesamiento disponibles | | |
| **Relación** | RU-04, RU-15, RU-16 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 49. Requisito funcional RF-06*

| | | | |
|---|---|---|---|
| **ID** | RF-07 | | |
| **Título** | El controlador podrá planificar sobresuscribiendo procesos en un mismo núcleo de procesamiento, hasta un máximo de dos procesos por núcleo | | |
| **Descripción** | El controlador tendrá la posibilidad de ejecutar las aplicaciones sobresuscribiendo procesos en un mismo núcleo de procesamiento, hasta un máximo de dos procesos por núcleo | | |
| **Relación** | RU-18 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 50. Requisito funcional RF-07*

| | | | |
|---|---|---|---|
| **ID** | RF-08 | | |
| **Título** | El controlador obtendrá la indicación de inicio de planificación por entrada estándar | | |
| **Descripción** | El controlador esperará para iniciar la planificación de las aplicaciones paralelas hasta que reciba la indicación de inicio de planificación por entrada estándar | | |
| **Relación** | No se aplica | | |
| **Prioridad** | Media | **Fuente** | Desarrollador |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 51. Requisito funcional RF-08*

| | | | |
|---|---|---|---|
| **ID** | RF-09 | | |
| **Título** | El controlador aplicará por defecto la planificación de aplicaciones paralelas secuencial | | |
| **Descripción** | Si no se especifica ninguna política de planificación se aplica la planificación secuencial por defecto | | |
| **Relación** | No se aplica | | |
| **Prioridad** | Baja | **Fuente** | Desarrollador |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Media | **Verificabilidad** | Alta |

*Tabla 52. Requisito funcional RF-09*

| | | | |
|---|---|---|---|
| **ID** | RF-10 | | |
| **Título** | El controlador obtendrá la indicación de suspensión de la planificación por entrada estándar | | |
| **Descripción** | Si el controlador recibe una indicación de suspensión por entrada estándar detendrá la planificación de las futuras aplicaciones paralelas a planificar | | |
| **Relación** | RU-23 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 53. Requisito funcional RF-10*

| | | | |
|---|---|---|---|
| **ID** | RF-11 | | |
| **Título** | El controlador obtendrá la indicación de reanudación de la planificación por entrada estándar | | |
| **Descripción** | Si el controlador recibe una indicación de reanudación por entrada estándar y la planificación se encuentra suspendida reanudará la planificación de las futuras aplicaciones paralelas a planificar | | |
| **Relación** | RU-24 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 54. Requisito funcional RF-11*

| | | | |
|---|---|---|---|
| **ID** | RF-12 | | |
| **Título** | Las aplicaciones paralelas estarán compuestas por el número de procesos que indique la carga de trabajo al iniciar su ejecución | | |
| **Descripción** | Las aplicaciones paralelas se dividirán en el número de procesos indicado por la carga de trabajo al comienzo de su ejecución | | |
| **Relación** | RU-04, RU-05 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 55. Requisito funcional RF-12*

| | | | |
|---|---|---|---|
| **ID** | RF-13 | | |
| **Título** | El controlador monitorizará el estado de los núcleos de procesamiento del sistema | | |
| **Descripción** | El controlador obtendrá el estado, libre o ejecutando procesos, de cada uno de los núcleos del sistema | | |
| **Relación** | RU-08, RU-16, RU-17, RU-18 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 56. Requisito funcional RF-13*

| | | | |
|---|---|---|---|
| **ID** | RF-14 | | |
| **Título** | El controlador comenzará la ejecución de una aplicación, teniendo en cuenta el estado de los núcleos de procesamiento, el número de procesos de la aplicación paralela y la política de planificación en aplicación | | |
| **Descripción** | El controlador calculará si es posible la ejecución de una aplicación paralela teniendo en cuenta los factores asociados | | |
| **Relación** | RU-05, RU-09, RU-13, RU-14, RU-17, RU-18 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 57. Requisito funcional RF-14*

| | | | |
|---|---|---|---|
| **ID** | RF-15 | | |
| **Título** | El controlador se comunicará remotamente con una aplicación paralela al inicio de su ejecución para activar la monitorización de la aplicación | | |
| **Descripción** | Las aplicaciones paralelas se dividirán en el número de procesos indicado por la carga de trabajo al comienzo de su ejecución | | |
| **Relación** | RU-07 | | |
| **Prioridad** | Alta | **Fuente** | Desarrollador |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 58. Requisito funcional RF-15*

| | | | |
|---|---|---|---|
| **ID** | RF-16 | | |
| **Título** | La aplicación paralela se comunicará remotamente con el controlador enviando los datos de monitorización durante su ejecución | | |
| **Descripción** | La aplicación paralela enviará remotamente sus datos de monitorización al controlador | | |
| **Relación** | RU-07, RU-10 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 59. Requisito funcional RF-16*

| | | | |
|---|---|---|---|
| **ID** | RF-17 | | |
| **Título** | El controlador almacenará los diez resultados monitorizados más recientes de las aplicaciones paralelas | | |
| **Descripción** | La aplicación paralela enviará remotamente sus datos de monitorización al controlador | | |
| **Relación** | RU-10, RU-11 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 60. Requisito funcional RF-17*

| | | | |
|---|---|---|---|
| **ID** | RF-18 | | |
| **Título** | La aplicación paralela se comunicará remotamente con el controlador para indicar su finalización | | |
| **Descripción** | La aplicación paralela enviará remotamente la indicación de que ha terminado su ejecución | | |
| **Relación** | RU-07, RU-12 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 61. Requisito funcional RF-18*

| | | | |
|---|---|---|---|
| **ID** | RF-19 | | |
| **Título** | El controlador incrementará el número de procesos asociados a una aplicación paralela en ejecución teniendo en cuenta el estado de los núcleos de procesamiento, la política de planificación aplicada y la situación de la planificación | | |
| **Descripción** | El controlador calculará si es posible incrementar el número de procesos de alguna de las aplicaciones paralelas en ejecución | | |
| **Relación** | RU-07, RU-15, RU-16 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 62. Requisito funcional RF-19*

| **ID** | RF-20 | | |
|---|---|---|---|
| **Título** | El controlador se comunicará remotamente con las aplicaciones paralelas para indicar el incremento en sus números de procesos | | |
| **Descripción** | El controlador enviará indicación de cuántos procesos debe crecer una aplicación paralela en el caso de aplicar maleabilidad | | |
| **Relación** | RU-07, RU-15, RU-16 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 63. Requisito funcional RF-20*

| **ID** | RF-21 | | |
|---|---|---|---|
| **Título** | El controlador se comunicará remotamente con las aplicaciones paralelas para indicar el núcleo del sistema en el que deben ejecutar cada uno de sus procesos | | |
| **Descripción** | El controlador enviará remotamente el mapeo de procesos-núcleos que debe llevar a cabo una aplicación paralela | | |
| **Relación** | RU-07, RU-17 | | |
| **Prioridad** | Alta | **Fuente** | Desarrollador |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 64. Requisito funcional RF-21*

| **ID** | RF-22 | | |
|---|---|---|---|
| **Título** | El controlador se comunicará remotamente con las aplicaciones paralelas para forzar su terminación | | |
| **Descripción** | El controlador tendrá la posibilidad de forzar la terminación de una aplicación remotamente | | |
| **Relación** | RU-07, RU-25 | | |
| **Prioridad** | Media | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 65. Requisito funcional RF-22*

| **ID** | RF-23 | | |
|---|---|---|---|
| **Título** | El controlador forzará la terminación de las aplicaciones paralelas del sistema ante un error grave | | |
| **Descripción** | El controlador terminará la ejecución de las aplicaciones que estén ejecutando si se detecta un error grave en el sistema | | |
| **Relación** | RU-25 | | |
| **Prioridad** | Media | **Fuente** | Desarrollador |
| **Necesidad** | Deseable | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 66. Requisito funcional RF-23*

| | | | |
|---|---|---|---|
| **ID** | RF-24 | | |
| **Título** | El controlador mostrará mensajes de error por salida estándar | | |
| **Descripción** | El controlador indicará el error producido cuando se detecte | | |
| **Relación** | RU-26 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 67. Requisito funcional RF-24*

| | | | |
|---|---|---|---|
| **ID** | RF-25 | | |
| **Título** | El controlador mostrará mensajes de información de la inicialización del sistema por salida estándar | | |
| **Descripción** | El controlador indicará el estado de la inicialización del sistema por salida estándar | | |
| **Relación** | RU-27 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 68. Requisito funcional RF-25*

| | | | |
|---|---|---|---|
| **ID** | RF-26 | | |
| **Título** | El controlador mostrará mensajes de información de la posibilidad de introducir comandos por salida estándar | | |
| **Descripción** | El controlador indicará vía salida estándar que puede aceptar la introducción de comandos a través de la entrada estándar | | |
| **Relación** | RU-28 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 69. Requisito funcional RF-26*

| | | | |
|---|---|---|---|
| **ID** | RF-27 | | |
| **Título** | El controlador mostrará mensajes de información de la suspensión del sistema por salida estándar | | |
| **Descripción** | El controlador indicará vía salida estándar la suspensión del sistema cuando se produzca | | |
| **Relación** | RU-31 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 70. Requisito funcional RF-27*

| | | | |
|---|---|---|---|
| **ID** | RF-28 | | |
| **Título** | El controlador mostrará mensajes de información de la reanudación del sistema por salida estándar | | |
| **Descripción** | El controlador indicará vía salida estándar la reanudación del sistema cuando se produzca | | |
| **Relación** | RU-07, RU-32 | | |
| **Prioridad** | Baja | **Fuente** | Cliente |
| **Necesidad** | Opcional | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 71. Requisito funcional RF-28*

| | | | |
|---|---|---|---|
| **ID** | RF-29 | | |
| **Título** | El controlador mostrará mensajes de información del inicio de una aplicación paralela por salida estándar | | |
| **Descripción** | El controlador indicará vía salida estándar el inicio de la ejecución de una aplicación paralela del sistema cuando se produzca | | |
| **Relación** | RU-33 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 72. Requisito funcional RF-29*

| | | | |
|---|---|---|---|
| **ID** | RF-30 | | |
| **Título** | El controlador mostrará mensajes de información de la finalización de una aplicación paralela por salida estándar | | |
| **Descripción** | El controlador indicará vía salida estándar la finalización de la ejecución de una aplicación paralela del sistema cuando se produzca | | |
| **Relación** | RU-34 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 73. Requisito funcional RF-30*

| | | | |
|---|---|---|---|
| **ID** | RF-31 | | |
| **Título** | El controlador mostrará mensajes de información del incremento de procesos de una aplicación paralela por salida estándar | | |
| **Descripción** | El controlador indicará vía salida estándar el incremento de procesos de una aplicación paralela del sistema cuando se produzca | | |
| **Relación** | RU-35 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 74. Requisito funcional RF-31*

| | | | |
|---|---|---|---|
| **ID** | RF-32 | | |
| **Título** | El controlador mostrará mensajes de información del mapeo entre procesos y núcleos de procesamiento del sistema por salida estándar | | |
| **Descripción** | El controlador indicará vía salida estándar mensajes de información del mapeo entre procesos y núcleos de procesamiento del sistema, cuando se produzcan un cambio en el estado de los mismos | | |
| **Relación** | RU-36 | | |
| **Prioridad** | Alta | **Fuente** | Desarrollador |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 75. Requisito funcional RU-38*

| | | | |
|---|---|---|---|
| **ID** | RF-33 | | |
| **Título** | El controlador mostrará mensajes de información de la finalización de la planificación del sistema completo por salida estándar | | |
| **Descripción** | El controlador indicará vía salida estándar la finalización de la planificación del sistema completo, cuando se produzca | | |
| **Relación** | RU-37 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 76. Requisito funcional RF-33*

| | | | |
|---|---|---|---|
| **ID** | RF-34 | | |
| **Título** | El controlador permitirá la finalización del sistema por entrada estándar | | |
| **Descripción** | El controlador permitirá finalizar el sistema completo con la indicación de finalización del sistema | | |
| **Relación** | No se aplica | | |
| **Prioridad** | Baja | **Fuente** | Desarrollador |
| **Necesidad** | Opcional | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 77. Requisito funcional RF-34*

#### 4.2.1.2 Requisitos no funcionales

A continuación, se presentan los requisitos no funcionales recogidos para el presente proyecto

| | | | |
|---|---|---|---|
| **ID** | RNF-01 | | |
| **Título** | El sistema tendrá aplicaciones paralelas basadas en *benchmarks* iterativos de uso de procesamiento | | |
| **Descripción** | Las aplicaciones paralelas intensivas en procesamiento del sistema consistirán en *benchmarks* con procesamiento predominante | | |
| **Relación** | RU-01 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 78. Requisito no funcional RNF-01*

| | | | |
|---|---|---|---|
| **ID** | RNF-02 | | |
| **Título** | El sistema tendrá aplicaciones paralelas basadas en *benchmarks* iterativos de acceso a memoria | | |
| **Descripción** | Las aplicaciones paralelas intensivas en memoria del sistema consistirán en *benchmarks* intensivos en el acceso a memoria | | |
| **Relación** | RU-02 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 79. Requisito no funcional RNF-02*

| | | | |
|---|---|---|---|
| **ID** | RNF-03 | | |
| **Título** | El sistema tendrá aplicaciones paralelas basadas en *benchmarks* iterativos de uso de rutinas de comunicación | | |
| **Descripción** | Las aplicaciones paralelas intensivas en comunicación del sistema consistirán en *benchmarks* con el uso de rutinas de comunicación predominante | | |
| **Relación** | RU-03 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 80. Requisito no funcional RNF-03*

| | | | |
|---|---|---|---|
| **ID** | RNF-04 | | |
| **Título** | El sistema tendrá una carga de trabajo estática que indique el número de aplicaciones, su tipo y el número de procesos | | |
| **Descripción** | El sistema poseerá una carga de trabajo estática que permita conocer el número total de aplicaciones, el tipo de las aplicaciones y el número de procesos que las forman | | |
| **Relación** | RU-04 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 81. Requisito no funcional RNF-04*

| | | | |
|---|---|---|---|
| **ID** | RNF-05 | | |
| **Título** | El sistema tendrá una infraestructura de control que atienda la carga de trabajo y la planifique | | |
| **Descripción** | El sistema poseerá un controlador capaz de planificar futuras aplicaciones a ejecutar aplicando las distintas políticas de planificación | | |
| **Relación** | RU-04 | | |
| **Prioridad** | Alta | **Fuente** | Cliente |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 82. Requisito no funcional RNF-05*

| | | | |
|---|---|---|---|
| **ID** | RNF-06 | | |
| **Título** | Los datos a monitorizar de una aplicación paralela serán: tiempo de ejecución, tiempo de ejecución en modo usuario, tiempo de rutinas de comunicación, megaflops y datos de contador *hardware*, todo ello por iteración | | |
| **Descripción** | Los datos que el controlador recibe por parte de la aplicación como parte de su monitorización son los indicados | | |
| **Relación** | No se aplica | | |
| **Prioridad** | Alta | **Fuente** | Desarrollador |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 83. Requisito no funcional RNF-06*

| | | | |
|---|---|---|---|
| **ID** | RNF-07 | | |
| **Título** | Los datos de estado de una aplicación paralela serán monitorizados utilizando la biblioteca externa *PAPI* | | |
| **Descripción** | La medida de los datos a monitorizar se realizará con la biblioteca *PAPI* | | |
| **Relación** | No se aplica | | |
| **Prioridad** | Alta | **Fuente** | Desarrollador |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 84. Requisito no funcional RNF-07*

| | | | |
|---|---|---|---|
| **ID** | RNF-08 | | |
| **Título** | El sistema estará integrado con la última versión de FlexMPI | | |
| **Descripción** | Las aplicaciones deben hacer uso de FlexMPI para utilizar sus funcionalidades de paralelismo | | |
| **Relación** | No se aplica | | |
| **Prioridad** | Alta | **Fuente** | Desarrollador |
| **Necesidad** | Esencial | **Claridad** | Alta |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 85. Requisito no funcional RNF-08*

| | | | |
|---|---|---|---|
| **ID** | RNF-09 | | |
| **Título** | El sistema debe ser compatible con una arquitectura de memoria distribuida | | |
| **Descripción** | Las funcionalidades del sistema deben ser aplicables a una arquitectura de memoria distribuida | | |
| **Relación** | No se aplica | | |
| **Prioridad** | Alta | **Fuente** | Desarrollador |
| **Necesidad** | Esencial | **Claridad** | Media |
| **Estabilidad** | Alta | **Verificabilidad** | Alta |

*Tabla 86. Requisito no funcional RNF-09*

## 4.3 Análisis de casos de uso

Los casos de uso presentan la interacción del usuario con el sistema. La Ilustración 17 muestra un diagrama UML de los diferentes casos de uso recogidos en presente proyecto.

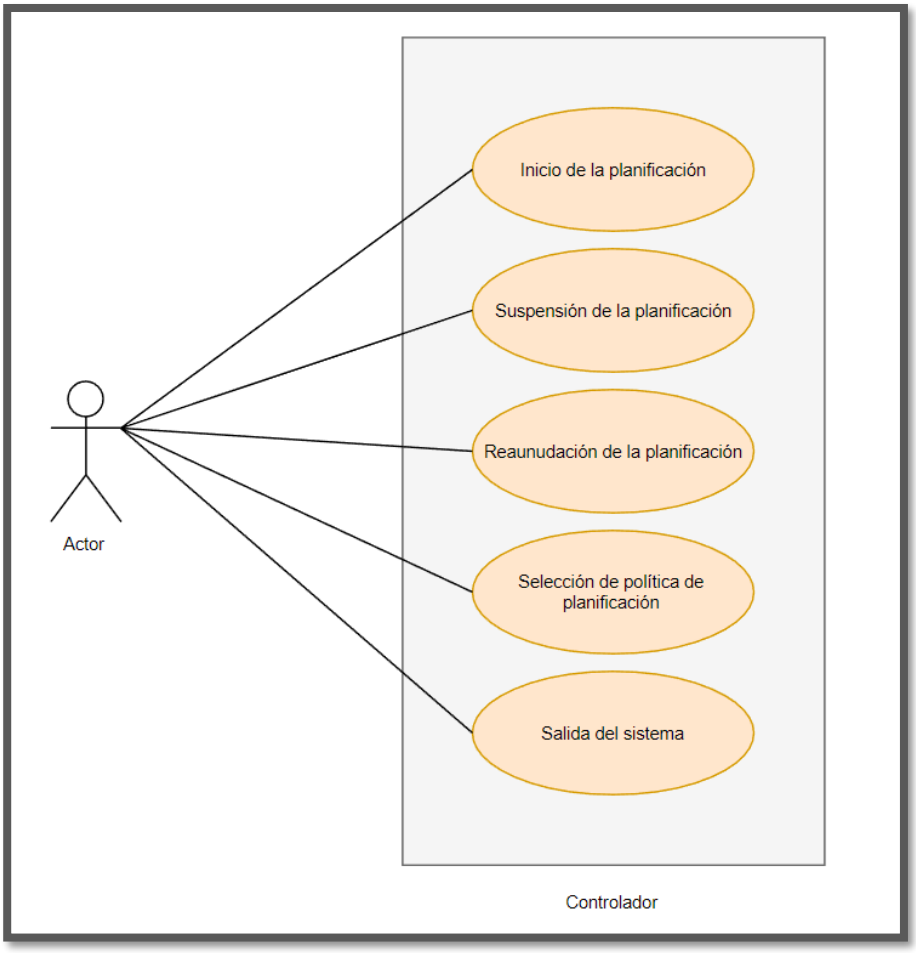



*Ilustración 17. Casos de uso para el sistema controlador*

Cada caso de uso se presentará de manera tabular. La Tabla 87 muestra la plantilla que seguirán los casos de uso expuestos.

| | |
|---|---|
| **ID** | CU-X |
| **Objetivo** | Objetivo del caso de uso |
| **Precondiciones** | Condiciones anteriores |
| **Postcondiciones** | Condiciones posteriores |
| **Escenario** | Descripción |

*Tabla 87. Plantilla tabular para la presentación de los casos de uso*

Los atributos que definen un caso de uso son los siguientes:

- **ID:** Código que identifica de forma unívoca a cada caso de uso (CU significa caso de uso y X un número de un dígito).
- **Objetivo:** Objetivo perseguido con el caso de uso.
- **Precondiciones:** Condiciones necesarias antes de darse el caso de uso.
- **Postcondiciones:** Condiciones necesarias después de darse el caso de uso.
- **Escenario:** Descripción más extensa del ámbito del caso de uso.

| | |
|---|---|
| **ID** | CU-1 |
| **Objetivo** | Iniciar el sistema de planificación |
| **Precondiciones** | El sistema no debe estar previamente iniciado |
| **Postcondiciones** | El sistema de planificación se ha iniciado con la política de planificación escogida |
| **Escenario** | Si el sistema no ha sido iniciado el usuario indica por entrada estándar el arranque del sistema. El sistema de planificación se inicia y ejecuta las aplicaciones conforme a la política de planificación. |

*Tabla 88. Caso de uso CU-1*

| | |
|---|---|
| **ID** | CU-2 |
| **Objetivo** | Suspender el sistema de planificación |
| **Precondiciones** | El sistema de planificación debe estar iniciado y en ejecución |
| **Postcondiciones** | El sistema de planificación ha quedado suspendido |
| **Escenario** | Si el sistema ha sido arrancado y se encuentra en ejecución, el usuario indica por entrada estándar la suspensión. El sistema de planificación detiene la planificación y se queda suspendido a la espera de reanudarse |

*Tabla 89. Caso de uso CU-2*

| | |
|---|---|
| **ID** | CU-3 |
| **Objetivo** | Reanudar el sistema de planificación |
| **Precondiciones** | El sistema de planificación debe estar iniciado y en suspensión |
| **Postcondiciones** | El sistema de planificación es reanudado |
| **Escenario** | Si el sistema ha sido arrancado y se encuentra en suspensión, el usuario indica por entrada estándar la reanudación. El sistema de planificación reanuda la planificación |

*Tabla 90. Caso de uso CU-3*

| | |
|---|---|
| **ID** | CU-4 |
| **Objetivo** | Seleccionar la política de planificación |
| **Precondiciones** | Ninguna |
| **Postcondiciones** | La política de planificación indicada queda seleccionada para aplicarse en el sistema |
| **Escenario** | El usuario indica por entrada estándar la política deseada. El sistema selecciona dicha política que determinará la planificación de las futuras aplicaciones |

*Tabla 91. Caso de uso CU-4*

| | |
|---|---|
| **ID** | CU-5 |
| **Objetivo** | Salida del sistema |
| **Precondiciones** | Ninguna |
| **Postcondiciones** | Se efectúa la salida del sistema y se detiene la ejecución del controlador |
| **Escenario** | El usuario indica por entrada estándar la salida del sistema. Se liberan los recursos empleados y se detiene la ejecución del controlador. |

*Tabla 92. Caso de uso CU-5*

## 4.4 Diseño del sistema

En este apartado se presenta el diseño realizado para lograr los objetivos expuestos y cumplir con los requisitos especificados. Se definirán con detalle los componentes que forman el sistema y se presenta las funcionalidades del mismo en su entorno operacional.

### 4.4.1 Nuevos componentes en FlexMPI/Controlador central

FlexMPI es una extensión de MPI y está implementada como una biblioteca a un nivel más alto de abstracción, que envuelve la implementación MPICH en su versión 3.2, capaz de monitorizar y predecir el rendimiento de una aplicación paralela y de distribuir la carga de trabajo en un sistema distribuido en función de las indicaciones que se le entreguen, tal y como se detalla en la sección 2.2 MPI y FlexMPI: soporte para la maleabilidad.

Se compone de una estructura modular en la que cada módulo se encarga de realizar una funcionalidad específica [41]. La Ilustración 18 presenta un esquema de dicha arquitectura modular de ejecución.

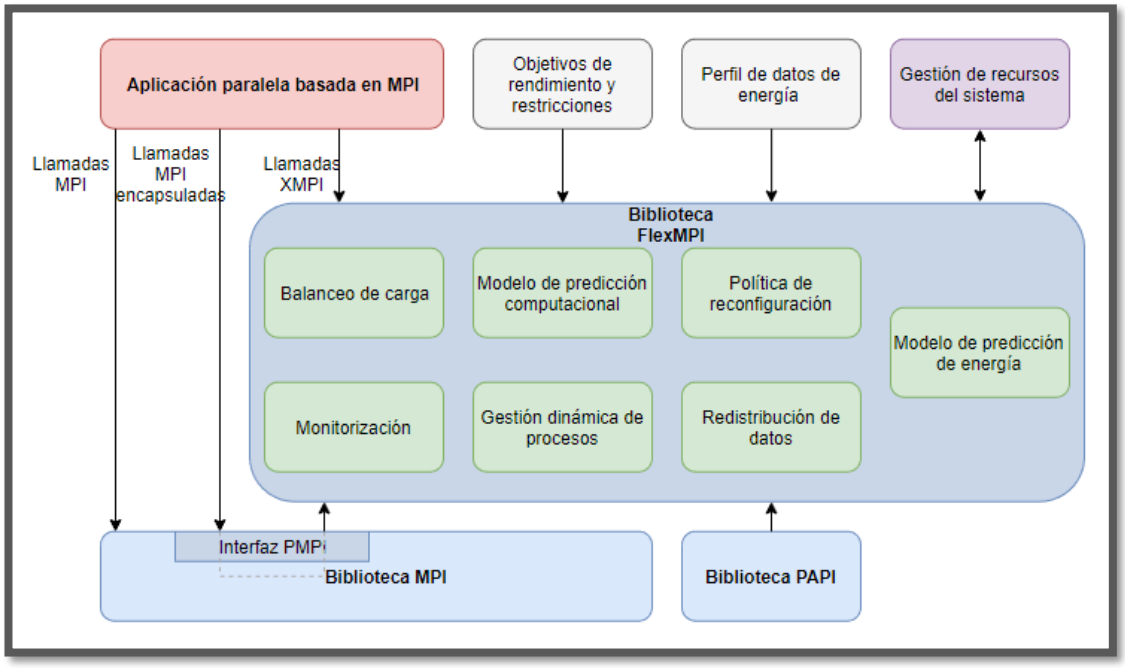



*Ilustración 18. Arquitectura modular de ejecución de FlexMPI*

Las aplicaciones paralelas utilizan funciones nativas y encapsuladas de MPI, a través de las cuales se realizan invocaciones a la biblioteca de FlexMPI y funciones directamente de FlexMPI. Cabe destacar el módulo de monitorización de FlexMPI utilizado para obtener las diferentess métricas de monitoricación de las aplicaciones. Dicho módulo utiliza la interfaz de bajo nivel de la biblioteca de PAPI para obtener los datos a través de contadores hardware.

Para hacer posible la conexión entre las apliciones paralelas y el controlador a diseñar, la comunicación entre las mismas se realiza a través de FlexMPI. Para ello se ha diseñado un nuevo componente modular en la biblioteca que permite

la comunicación con el controlador central. La Ilustración 19 presenta la arquitectura del sistema tras el cambio.

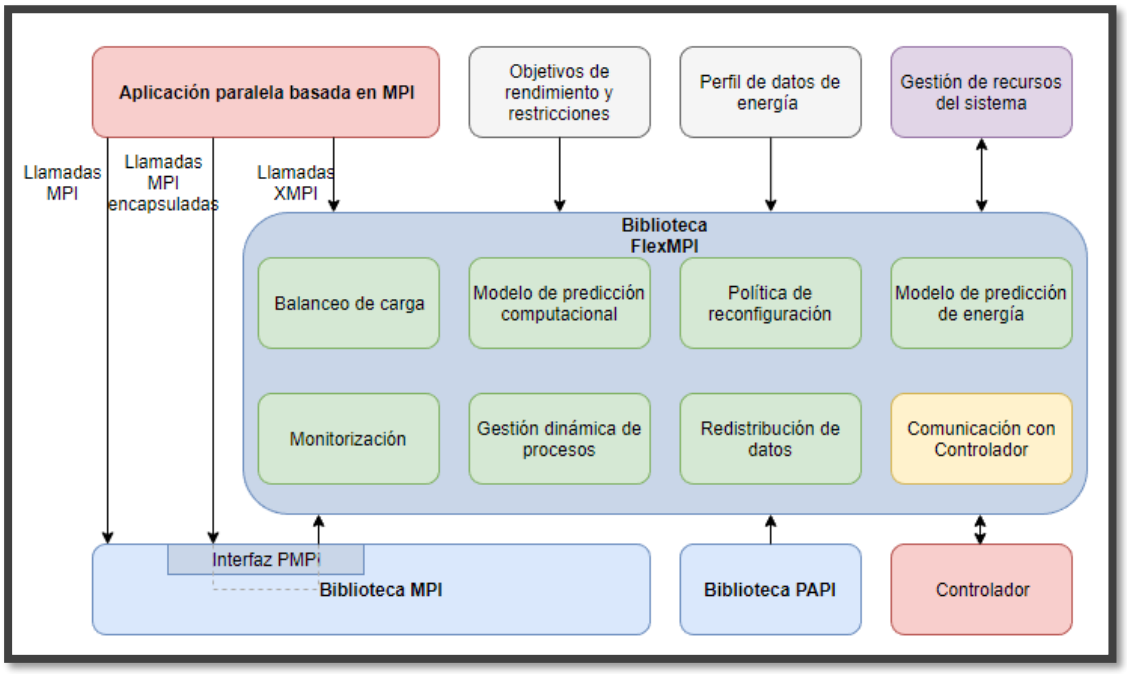



*Ilustración 19. Adición de módulo en FlexMPI de comunicación con el controlador central*

Dicho módulo de comunicación se compone de un hilo que se comunica con el controlador remotamente a través de puertos. Cada aplicación contará con hilo de comunicación con el controlador, puesto que cada aplicación cuenta con su instancia de la biblioteca. La Ilustración 20 ofrece una explicación más detallada del protocolo de comunicación que utiliza el sistema.

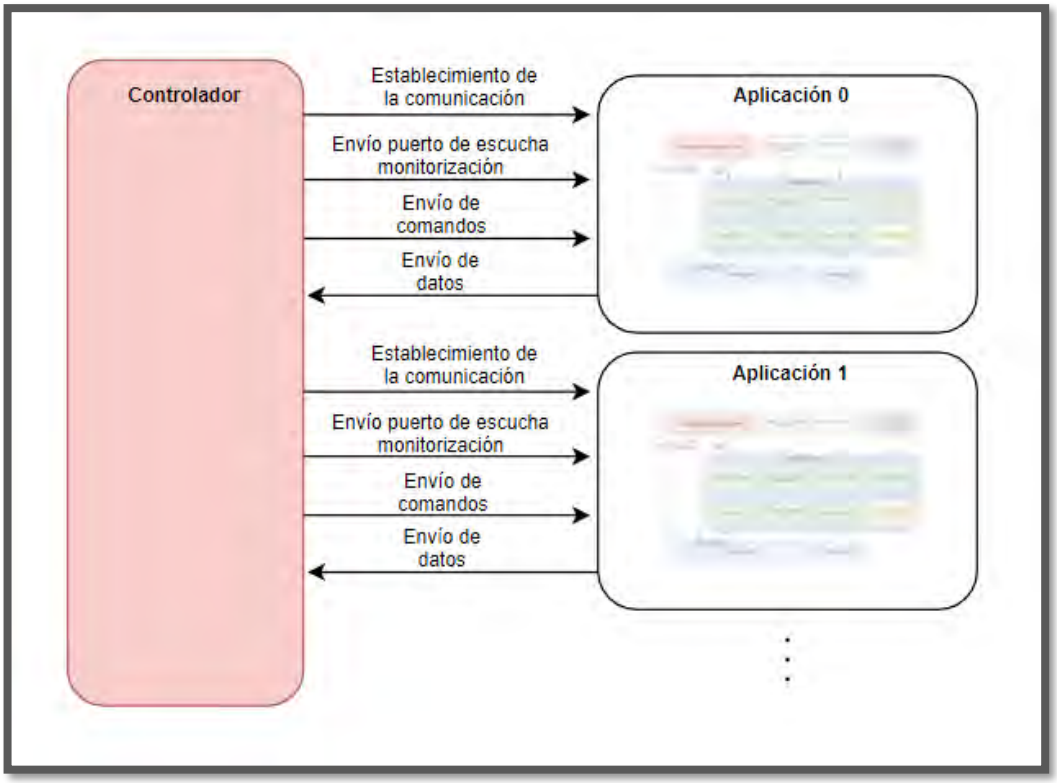



*Ilustración 20. Protocolo de comunicación entre aplicaciones paralelas y controlador*

Cuando una aplicación es desplegada, el controlador utiliza el módulo de comunicación de la biblioteca de FlexMPI para establecer la comunicación con la aplicación a través de un puerto conocido. Acto seguido, el controlador envía a la aplicación el puerto a través del cual realizará la escucha de los mensajes que la aplicación enviará. El controlador podrá enviar comandos de instrucción a la aplicación continuamente. Por último, la aplicación paralela enviará los datos de monitorización y finalización al controlador a través del puerto de escucha recibido.

El controlador consiste en una infraestructura central que administra la ejecución de las distintas aplicaciones paralelas existentes en el sistema. Se encarga de inicializar todos los recursos necesarios para la correcta ejecución de las diferentes tareas y de tomar las decisiones pertinentes en la gestión de las aplicaciones del sistema.

Su función principal es la de planificar la ejecución de las diferentes aplicaciones existentes en la carga de trabajo, decidiendo el orden en el que se ejecutan y los recursos que se asignan a las mismas. Para ello, necesita una infraestructura de *threads* que ayuden a realizar diversas tareas concurrentemente. La Ilustración 21 muestra el despliegue de procesos ligeros del controlador.

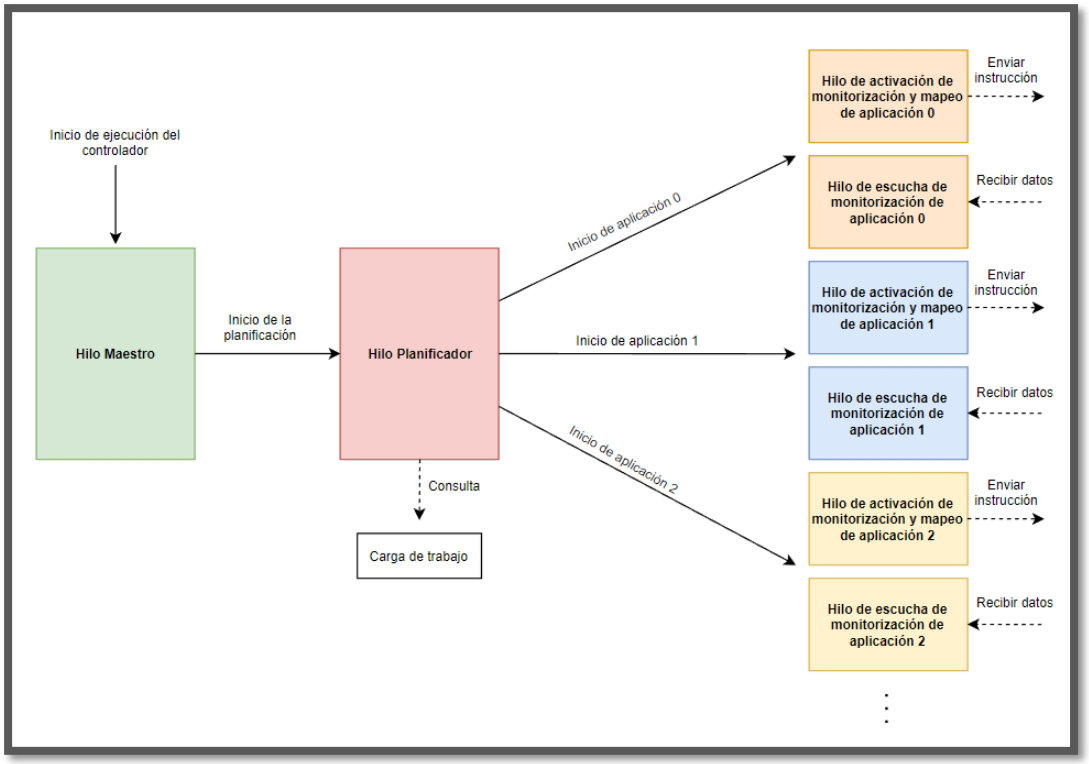



*Ilustración 21. Infraestructura de hilos del controlador*

En primer lugar, se inicia el flujo principal de ejecución del controlador que inicializa los recursos y estructuras de datos pertinentes. Este hilo maestro queda a la espera de la introducción de datos por entrada estándar, como la selección de la política de planificación que se desea, el inicio de la planificación, su suspensión, reanudación y la finalización del sistema. En el momento en el que el hilo maestro recibe la indicación de arrancar la planificación, ejecuta las aplicaciones paralelas remotamente conforme a la carga de trabajo, la política de planificación y los núcleos de procesamiento libres en el sistema. Para monitorizar el estado de las aplicaciones se crean dos hilos. El primero, comunica a la aplicación paralela el comienzo de la monitorización e indica sobre qué núcleos debe mapear sus procesos. A medida que las aplicaciones en ejecución terminan y dejan recursos libres, el hilo planificador determina las futuras aplicaciones a poner en ejecución.

Para realizar la gestión de las aplicaciones de la carga de trabajo, el diseño se apoya en dos elementos. La Ilustración 22 muestra esquemáticamente la composición de estos elementos.

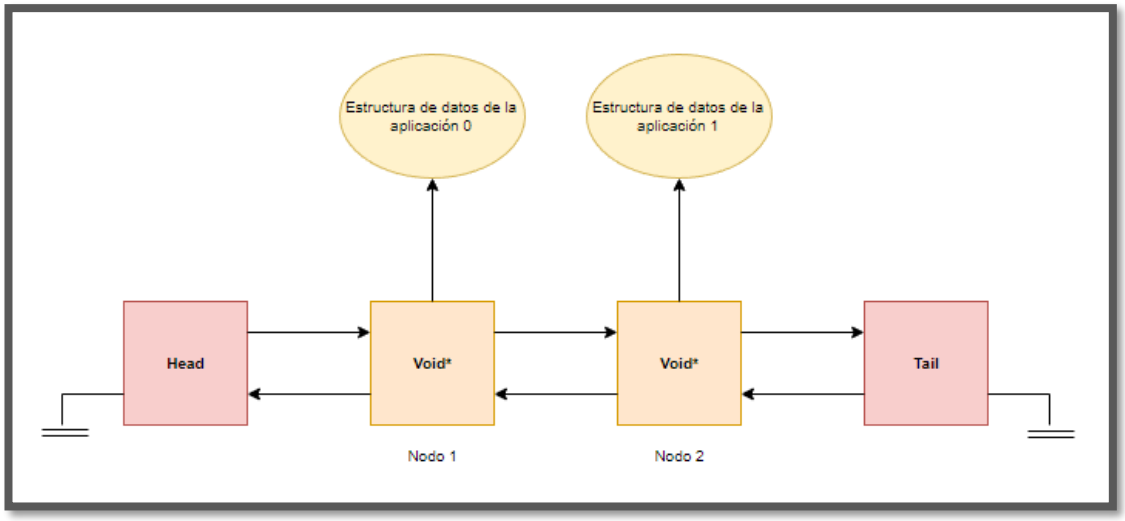



*Ilustración 22. Estructura de datos y lista doblemente enlazada para la gestión de aplicaciones*

El primer elemento consiste en una estructura que contiene datos relevantes de cada aplicación existente. Dicha estructura almacena la siguiente información:

- Número de nodos en los que se despliega la aplicación
- Nombre de cada nodo
- Número de procesos que forman la aplicación por nodo
- Puerto de escucha de la aplicación
- Puerto de envío de datos de la aplicación

Cabe destacar, que aunque los objetivos considerados en el presente proyecto se basan en el despliegue en un solo nodo del sistema, el diseño se ha realizado pensando en trabajos futuros en los que se pueda desplegar el sistema en más de un nodo.

El segundo elemento, consiste en una lista doblemente enlazada genérica que es capaz de registrar cualquier tipo de dato. Se ha diseñado así teniendo en cuenta la alta probabilidad de cambio que puede sufrir la manera en la que se describe una aplicación en el sistema.

Para determinar el estado de una aplicación paralela existente en el sistema se utilizan tres listas doblemente enlazadas que recogen a las aplicaciones con los siguientes estados:

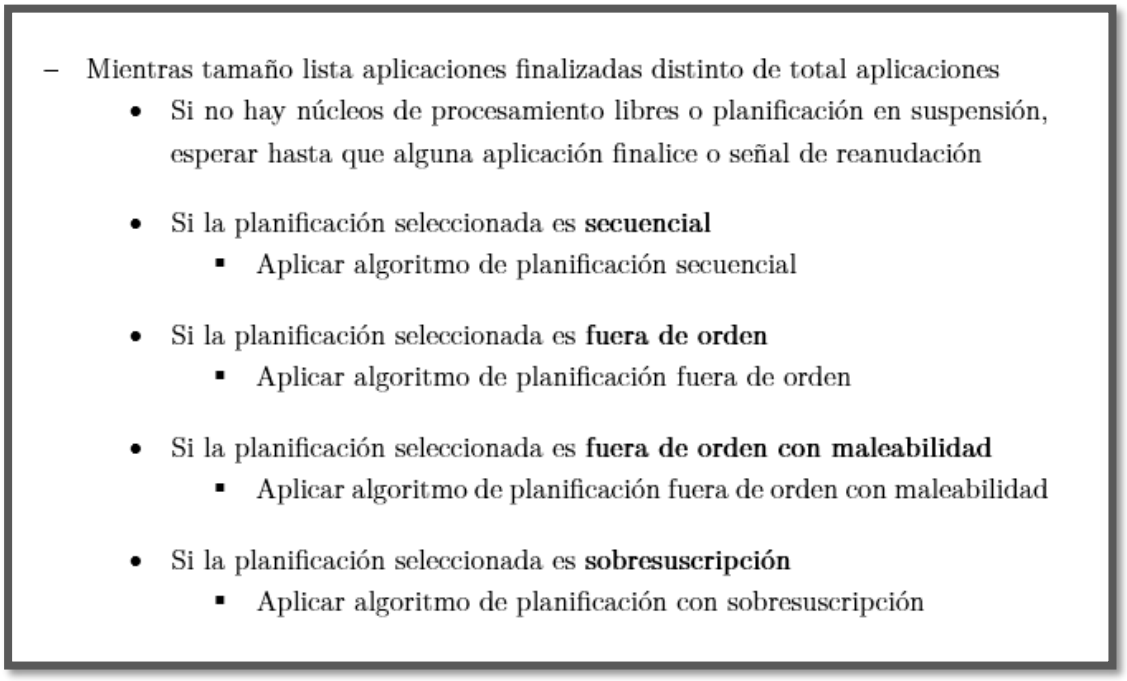
- Aplicaciones esperando para ser ejecutadas
- Aplicaciones en ejecución
- Aplicaciones finalizadas

La comunicación entre las aplicaciones y el controlador se realiza a través de puertos sockets. De esta manera, tanto si los procesos y controlador se sitúan

remotamente o de manera local en un mismo nodo, serán capaces de comunicarse entre sí, y siendo compatible tanto con un modelo de memoria distribuida como con un modelo de memoria compartida.

Para efectuar el inicio de las diferentes aplicaciones, se hace uso de comandos que son capaces de generar *scripts* de instrucciones en los nodos indicados (de manera remota o local) con los parámetros recogidos en la carga de trabajo.

El hilo planificador es el encargado de decidir el orden de ejecución de los procesos y cuestiones de maleabilidad de procesos y sobresuscripción, dependiendo del tipo de planificación en aplicación. La Ilustración 23 presenta un pseudocódigo que describe de manera muy simple su funcionamiento:



- Mientras tamaño lista aplicaciones finalizadas distinto de total aplicaciones
  - Si no hay núcleos de procesamiento libres o planificación en suspensión, esperar hasta que alguna aplicación finalice o señal de reanudación
  - Si la planificación seleccionada es **secuencial**
    - Aplicar algoritmo de planificación secuencial
  - Si la planificación seleccionada es **fuera de orden**
    - Aplicar algoritmo de planificación fuera de orden
  - Si la planificación seleccionada es **fuera de orden con maleabilidad**
    - Aplicar algoritmo de planificación fuera de orden con maleabilidad
  - Si la planificación seleccionada es **sobresuscripción**
    - Aplicar algoritmo de planificación con sobresuscripción

*Ilustración 23. Pseudocódigo del hilo planificador*

Los algoritmos de cada uno de los tipos de planificación se detallan en el apartado 4.4.2 Técnicas de planificación.

El hilo planificador realiza una espera pasiva, cuando ha aplicado sus funcionalidades, para no desperdiciar recursos, hasta que se produce un cambio en el estado del sistema.

Antes de ejecutar una aplicación se tiene en cuenta qué núcleos de procesamiento son los que se encuentran disponibles para que dicha aplicación ejecute sus procesos. Si la aplicación puede ejecutar, se elimina de la lista de aplicaciones esperando y se introduce en la lista de en ejecución. Además, se crea el hilo de

activación de la monitorización y mapeo, que indican a la aplicación que el controlador está preparado para recibir datos y por qué puerto los escuchará y sobre qué núcleos debe ejecutar sus procesos. Dicha indicación se envía a las aplicaciones tras realizar una espera de un tiempo definido como parámetro del sistema, que permite a las aplicaciones poder completar su despliegue en el inicio de su ejecución.

También se crea el hilo de escucha que se encarga de recibir los datos de monitorización que envía la aplicación al controlador y los almacena en un *buffer* circular disponible para cada aplicación. Su diseño permite almacenar un número de valores más recientes que se puede definir cómo parámetro del sistema. La Ilustración 24 presenta de manera esquemática la lógica de dicho *buffer* circular.

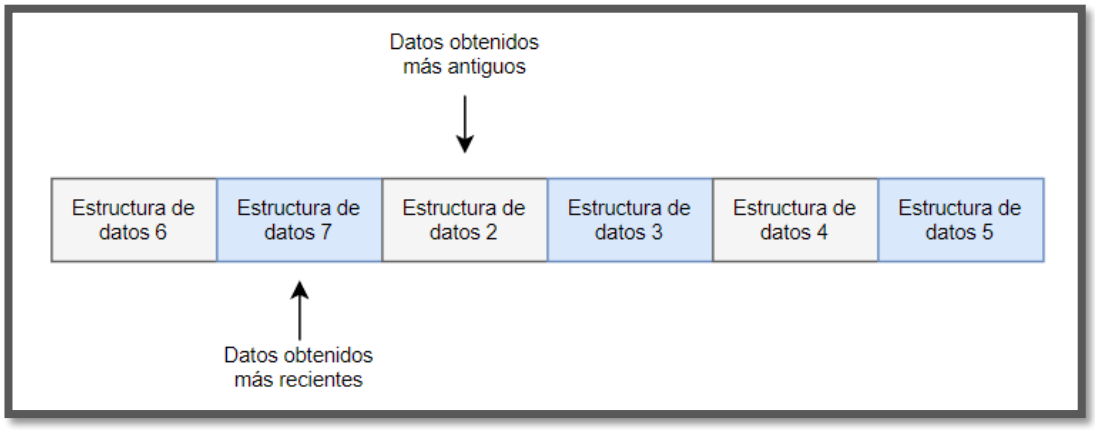



*Ilustración 24. Buffer circular para el almacenamiento de datos de monitorización*

Los datos de monitorización se almacenan como estructuras que contienen la siguiente información:

- Tiempo de ejecución por iteración
- Tiempo de ejecución en modo usuario por iteración
- Tiempo de uso de rutinas de comunicación por iteración
- Megaflops
- Valor de contador hardware

El valor de contador hardware se puede modificar y establecer conforme al criterio deseado en el sistema. Es capaz de informar del número de ciclos, accesos a memoria, accesos a caché, etc.

Por otra parte, los hilos de escucha también perciben la finalización de una aplicación. Tras dicho evento, se encargan de actualizar los valores pertinentes, liberando los núcleos de procesamiento que ocupaba la aplicación finalizada y

eliminando la aplicación de la lista de aplicaciones ejecutando para introducirla en la lista de aplicaciones finalizadas.

Todos los recursos que son compartidos entre hilos, son protegidos mediante técnicas de exclusión mutua, asegurando siempre un estado consistente de los valores con los que trabajan.

Cabe destacar, que los hilos activación de monitorización y mapeo y los hilos que escuchan los datos recibidos se crean y liberan a medida que los procesos inician su ejecución y finalizan, de manera que el sistema es escalable conforme a la demanda de aplicaciones y recursos disponibles.

Por otra parte, ante la detección de un error en el sistema o su terminación forzada, dichos eventos son capturados para liberar los recursos que han sido utilizados y comunicar a las aplicaciones que se encuentren ejecución que finalicen su desarrollo.

### 4.4.2 Técnicas de planificación

En este apartado se describen los algoritmos desarrollados con el objetivo de satisfacer las distintas políticas de planificación.

#### 4.4.2.1 Planificación secuencial

Este tipo de planificación es ampliamente utilizado en sistemas de procesamiento por lotes o *batch* y no supone esfuerzo desde el punto de vista algorítmico.

Se fundamenta en ejecutar las aplicaciones paralelas en el orden definido en la carga de trabajo, de manera que si una aplicación no puede ejecutarse por falta de núcleos de procesamiento libres, el planificador se detiene hasta que otra aplicación termina, para comprobar nuevamente si puede ponerla a ejecutar. La Ilustración 25 presenta un pseudocódigo del algoritmo.

- Mientras que el tamaño de la lista de aplicaciones esperando sea distinto de cero
  - Si hay recursos para ejecutar la siguiente aplicación de la lista de espera
    - Eliminar aplicación de la lista de espera
    - Introducir aplicación en la lista de aplicaciones en ejecución
    - Ejecutar aplicación
  - Si no hay recursos
    - Espera pasiva hasta que alguna aplicación en ejecución finalice

*Ilustración 25. Pseudocódigo para la planificación secuencial*

El algoritmo planificará las aplicaciones que estén esperando en la lista de aplicaciones en espera para la ejecución. Cuando hay recursos libres, elimina la aplicación de dicha lista, y la introduce en la lista de aplicaciones en ejecución. En el momento que no hay recursos para la aplicación considerada en ese momento, la planificación se detiene realizando una espera pasiva hasta que otra aplicación en ejecución termina. Las terminaciones de las aplicaciones son registradas por el hilo de escucha de cada aplicación. Dicho hilo es el encargado de eliminar las aplicaciones de la lista de ejecución e introducirlas en la lista de aplicaciones finalizadas.

La Ilustración 26 presenta un ejemplo del comportamiento del planificador secuencial, con una carga de trabajo de cuatro aplicaciones y un sistema de cuatro núcleos de procesamiento.

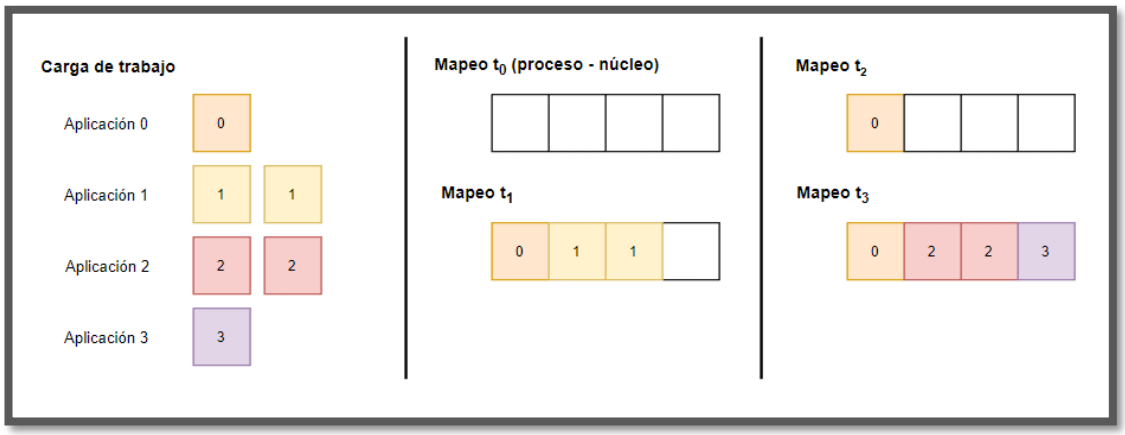



*Ilustración 26. Escenario de planificación secuencial*

La carga de trabajo define el orden de las aplicaciones que coincide con su valor numérico. Las aplicaciones 0 y 3 cuentan con un proceso, mientras que las aplicaicones 1 y 2 cuentan con dos procesos. En el estado inicial, los núcleos de procesamiento no ejecutan ningún proceso de ninguna aplicación paralela.

Cuando arranca el planificador, pone a ejecutar la aplición 0 y 1, pues hay recursos disponibles para ellas. Sin embargo, cuando llega a la aplicación 2, observa que no hay recursos disponibles y por tanto se bloquea a la espera de que alguna de las aplicaciones termine.

En este caso, termina primeramente la aplicación 1, y por tanto quedan dos núcleos de procesamiento libres. Es entonces, cuando la aplicación 2 puede ser ejecutada. Como también hay un núcleo libre, la aplicación 3 que solo dispone de un proceso, comienza su ejecución.

Como se puede observar, el principal inconveniente es que aunque existan núcleos de procesamiento libres que puedan ser ocupados por aplicaciones de la carga de trabajo, estos no se aprovechan al quedar el planificador bloqueado por alguna aplicación que en ese momento necesite más recursos. En el ejemplo planteado aunque la aplicación 3 podía ejecutar en $t_1$, aprovechando así todos los recursos del sistema y derivando en que la ejecución del sistema completo sea más rápida, ha tenido que esperar a que hubiese recursos suficientes para la aplicación que le precedía, y así que el planificador pudiera considerarla.

#### 4.4.2.2 Planificación fuera de orden

El presente tipo de planificación intenta mitigar el principal inconveniente que presentaba la planificación secuencial. La Ilustración 27 presenta el pseudocódigo del algoritmo.

- Mientras que el tamaño de la lista de aplicaciones esperando sea distinto de cero
  - Recorrer elemento a elmento toda la lista de aplicaciones esperando para ejecutar
    - Si aplicación actual no supera recursos libres
      - Eliminar aplicación de la lista de espera
      - Introducir aplicación en la lista de aplicaciones en ejecución
      - Ejecutar aplicación
  - Espera pasiva hasta que alguna aplicación finalice

*Ilustración 27. Pseudocódigo para la planificación fuera de orden*

Se basa en permitir ejecutar los procesos de la carga de trabajo fuera del orden especificado, siempre que existan recursos para su ejecución, incitando así al mayor aprovechamiento de los recursos en el momento actual en el que transcurre la planificación. La Ilustración 28 muestra el escenario anterior.

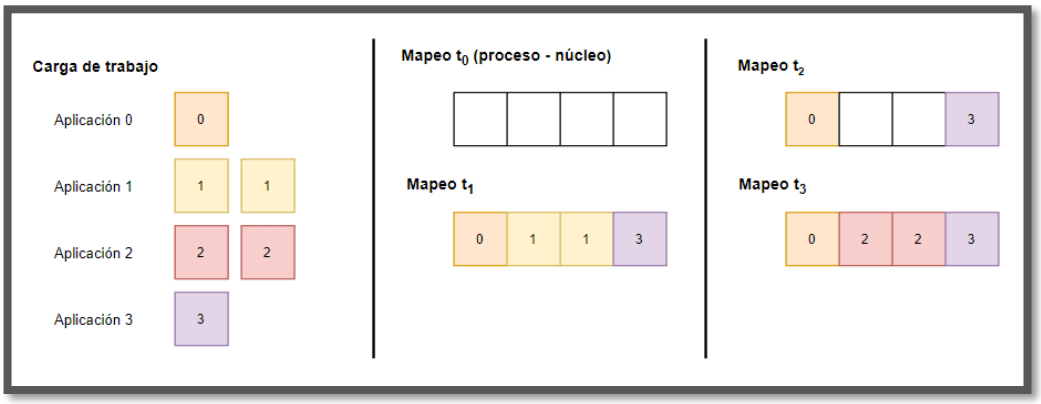



*Ilustración 28. Escenario 1 de planificación fuera de orden*

En esta ocasión el mapeo realizado en $t_{1,}$, permite la ejecución de la aplicación 3, aprovechándose todos los núcleos del sistema. Este escenario resulta especialmente favorable para esta política de planificación, pues al terminar la aplicación 1, se ejecuta la aplicación 2, sin haber desperciado núcleos de procesamiento.

Sin embargo, presenta el problema de que no aprovecha núcleos de procesamiento que no son especificados en la carga de trabajo, aunque dichos núcleos se encuentren ociosos durante una gran cantidad de tiempo. La Ilustración 29 presenta de manera gráfica este inconveniente.

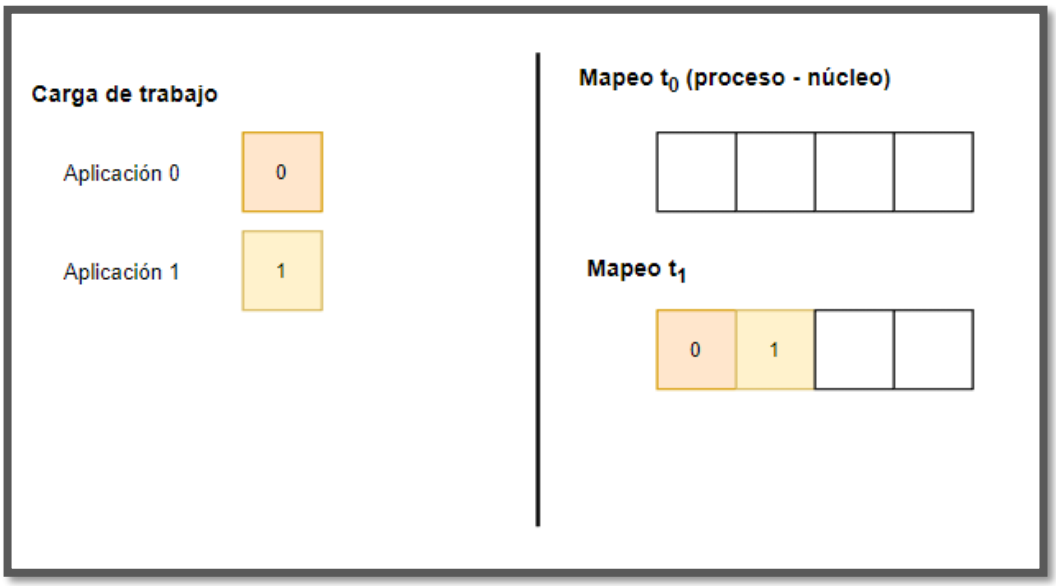



*Ilustración 29. Escenario 2 de planificación fuera de orden*

#### 4.4.2.3 Planificación fuera de orden con maleabilidad

Este tipo de planificación sigue las pautas en las que se basa la planificación fuera de orden pero añade la aplicación de maleabilidad de procesos.

Cabe destacar que en el presente sistema se considera que solo se puede incrementar el número de procesos de una aplicación y nunca dismunuirlo, pues se asume que el número de procesos que especifica la carga de trabajo supone el mínimo número de procesos que necesita o exige una aplicación.

Se fundamenta primeramente en realizar la planificación de procesos fuera de orden. Una vez aplicada, lo que quiere decir que en ese momento no existen más aplicaciones en la lista de aplicaciones esperando para ser ejecutadas cuyo mínimo número de procesos pueda ser abarcado, si quedan recursos libres, se realiza el algoritmo de maleabilidad.

Puesto que las aplicaciones intensivas en rutinas de comunicación no son escalables, ya que al contar con un mayor número de procesos tienen que realizar un mayor esfuerzo para realizar el intercambio de datos y mensajes, la maleabilidad no se aplica sobre este tipo de aplicaciones

La Ilustración 30 presenta el pseudocódigo del algoritmo.

- Mientras que el tamaño de la lista de aplicaciones esperando sea distinto de cero
  - Recorrer elemento a elmento toda la lista de aplicaciones esperando para ejecutar
    - Si aplicación actual no supera recursos libres
      - Eliminar aplicación de la lista de espera
      - Introducir aplicación en la lista de aplicaciones en ejecución
      - Ejecutar aplicación
  - Si todavía quedan recursos libres
    - Aplicar maleabilidad sobre las aplicaciones en ejecución, excepto las intensivas en comunicación
  - Espera pasiva hasta que alguna aplicación finalice

*Ilustración 30. Pseudocódigo para la planificación fuera de orden con maleabilidad*

El modelo de maleabilidad consiste en un algoritmo ávaro que intenta repartir los núcleos de procesamiento libres entre las aplicaciones de la lista de aplicaciones esperando para ejecutar, empezando por la aplicación más antigua insertada en la misma, equitativamente. La Ilustración 31 presenta el escenario en el que los recursos de la planificación fuera de orden quedaban desaprovechados.

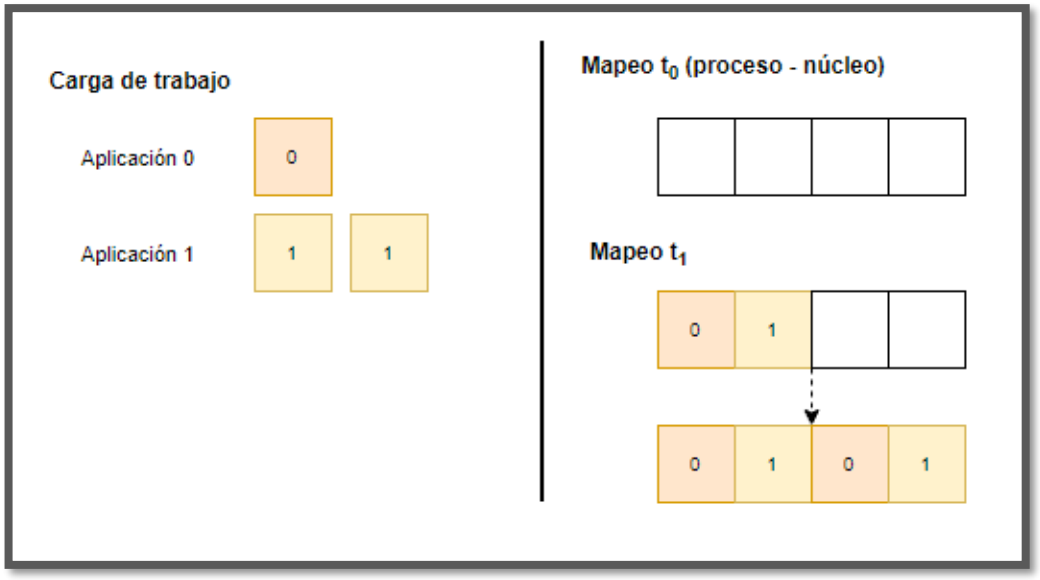



*Ilustración 31. Escenario 1 de planificación fuera de orden con maleabilidad*

La aplicación de maleabilidad permite aprovechar los recursos que quedan libres tras aplicar la planificación fuera de orden, cuando no se puede iniciar la ejecución de ninguna otra aplicación.

Para explicar su funcionamiento con mayor detalle, se propone un segundo escenario que cuenta con ocho núcleos y tres aplicaciones de uno, dos y cuatro procesos respectivamente, que no son intensivas en operaciones de comunicación. La Ilustración 32 presenta el escenario gráficamente.

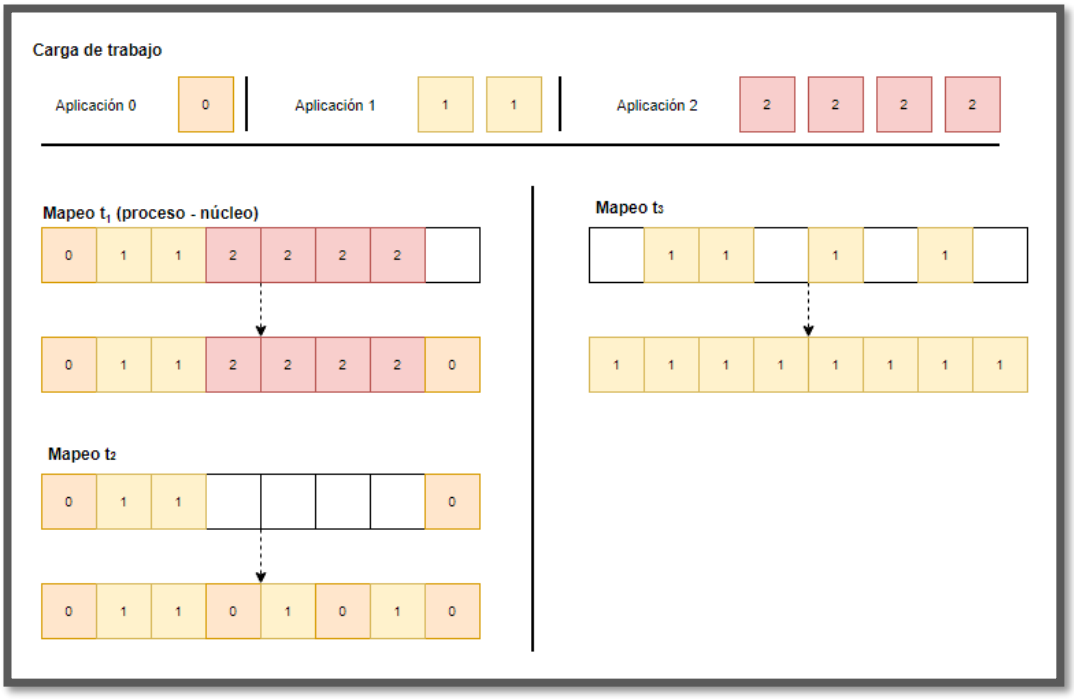



*Ilustración 32. Escenario 2 de planificación fuera de orden con maleabilidad*

En $t_1$ se produce el mapeo fuera de orden, que en este caso, por contar con los recursos pertinentes coincide con el orden especificado en la carga de trabajo. Puesto que el sistema no puede planificar más procesos, aplica maleabilidad, recorriendo la lista de aplicaciones esperando para ejecutar empezando por su entrada más antigua. De esta manera, se determina que la aplicación 0 puede crecer en un proceso.

En $t_2$ la aplicación 2 termina, y las aplicaciones 0 y 1 se pueden repartir los recursos libres, comenzando nuevamente por la entrada más antigua de la lista de aplicaciones esperando a ser ejecutados.

Finalmente, en $t_3$ la aplicación 0 termina, y la aplicación 1 puede contar con todos los recursos disponibles del sistema.

Este tipo de planificación puede presentar casos no deseados en ocasiones. En la Ilustración 33 se presenta gráficamente un escenario en el que se puede presentar una configuración de planificación no desada.

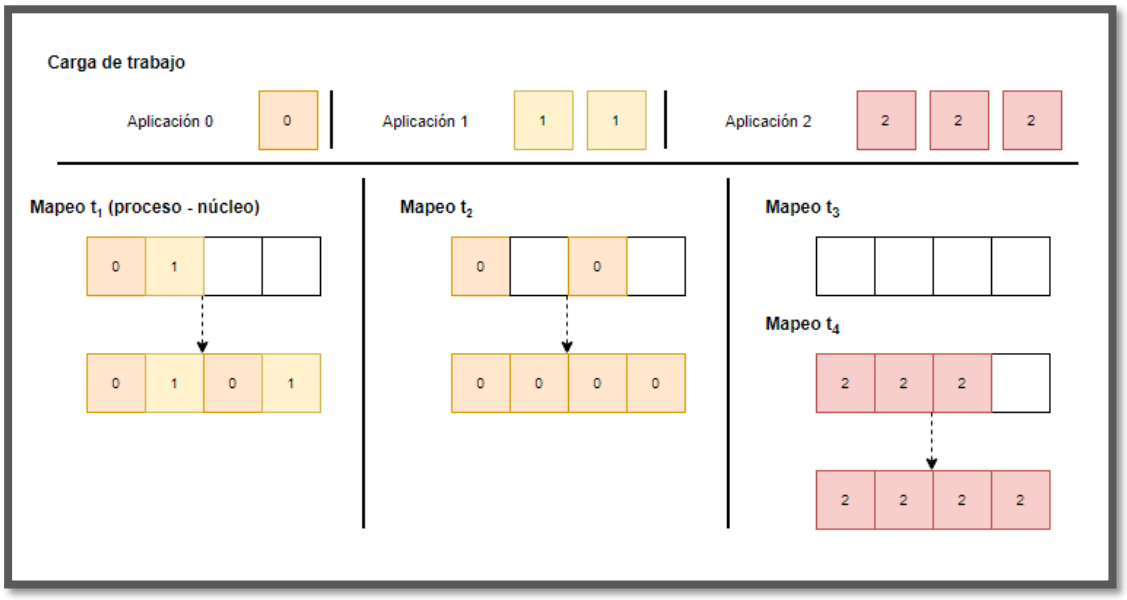



*Ilustración 33. Escenario 3 de planificación fuera de orden con maleabilidad*

En $t_1$ comienza la ejecución de las aplicaciones 0 y 1, mientras que la aplicación 2 debe esperar pues no se dispone de recursos para ella. Puesto que se ha aplicado la planificación fuera de orden y no se puede comenzar la ejecución de ningún otro proceso, se reparten los recursos sobrantes entra las aplicaciones 0 y 1.

Acto seguido, en $t_2$ termina la ejecución de la aplicación 1. Sin embargo, como la aplicación 0 fue incrementada anteriormente, no existen recursos suficientes para la aplicación 2, que existirían en caso de no haberse aplicado la maleabilidad. Por tanto, en vez de comenzar la ejecución de la aplicación 2, ésta debe esperar hasta que la aplicación 0 termine, que por no haber otras aplicaciones que puedan ejecutar, abarcará todos los recursos del sistema.

Finalmente, en $t_3$ la aplicación 0 termina, y en $t_4$ comenzará la ejecución de la aplicación 2, que por ser la única en la carga de trabajo sin ejecutar, abarcará todos los recursos del sistema.

El hecho de que la aplicación 2 no haya ejecutado antes, puede suponer la configuración óptima, si el sistema completo se ejecuta más rápido. Sin embargo, el hecho de que la aplicación 0 y 1 hayan incrementado sus números de procesos y posteriormente la aplicación 0 se haya incrementado nuevamente, puede haber supuesto la configuración óptima. Este comportamiento depende de la duración de cada aplicación, que en principio es distinta para cada una.

Por tanto, este tipo de planificación no aporta una configuración totalmente eficiente, como la intuición puede llevar a pensar, si no que sólo asegura que se aprovechan al máximo los recursos del sistema de una manera ávara.

La Ilustración 34 presenta un escenario en el se cuenta con una aplicación de comunicación en la carga de trabajo.

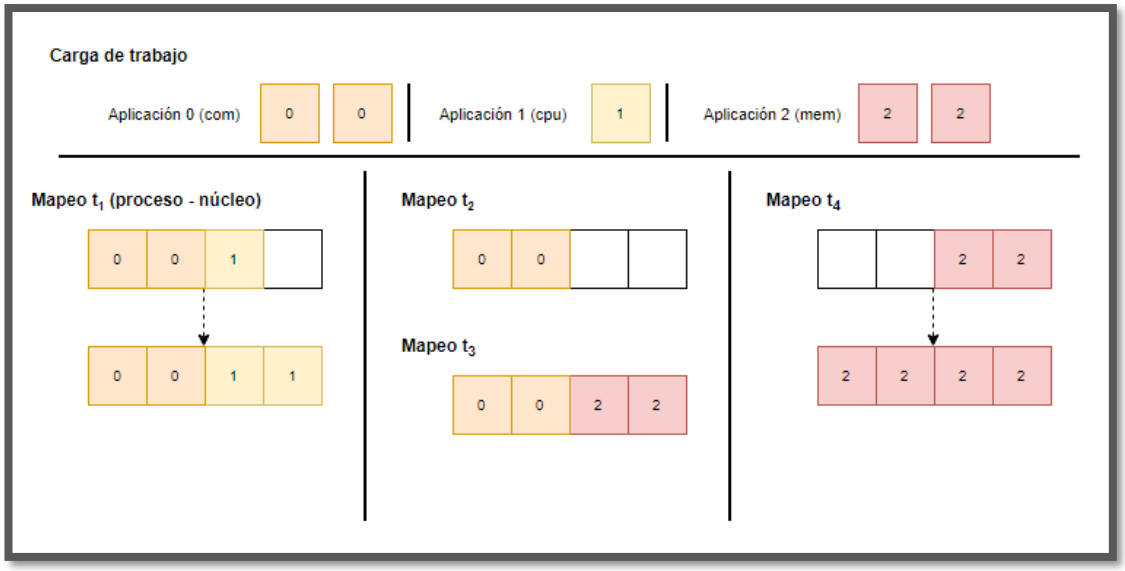



*Ilustración 34. Escenario 4 de planificación fuera de orden con maleabilidad*

En $t_1$, se asigna la aplicación 0 de comunicación a los dos primeros núcleos de procesamiento y la aplicación 1 al tercer núcleo. Aunque la aplicación 0 es la

primera en la lista de aplicaciones en ejecución, al ser intensiva en comunicación se determina que es contraproducente aumentar su número de procesos, y se pasa a aplicar maleabilidad sobre la siguiente aplicación en la lista de ejecución, que es la aplicación 1, intensiva en cpu, y por tanto se incrementa en un proceso. En $t_2$, la aplicación 1 termina, y nuevamente no se aplica maleabilidad sobre la aplicación 0. Como no hay más aplicaciones en ejecución se pasa a asignar la aplicación 2 a los núcleos libres. Cuando en $t_4$ la aplicación 0 finaliza su ejecución, la aplicación 2, intensiva en memoria, aumenta en dos el número de procesos asociados a la misma, al ser la única existente en el sistema en ese momento.

#### 4.4.2.4 Planificación con sobresuscripción

La planificación basada en sobresuscripción se basa en compartir un mismo núcleo entre varios procesos. La Ilustración 35 presenta el pseudocódigo del algoritmo.

- Mientras que el tamaño de la lista de aplicaciones esperando sea distinto de cero
  - Obtener primera aplicación de la lista de aplicaciones esperando para ejecutar.
  - Si hay recursos disponibles para su ejecución (teniendo en cuenta el número definido de procesos que se pueden sobresuscribir en un mismo núcleo y que las aplicaciones de comunicación no admiten sobresuscripción).
    - Eliminar aplicación de la lista de espera
    - Introducir aplicación en la lista de aplicaciones en ejecución
    - Aplicar modelo de sobresuscripción para obtener el mapeo más beneficioso para la aplicación, excepto en aplicaciones de comunciación
    - Ejecutar aplicación
  - Si no hay recursos disponibles, espera pasiva hasta que alguna aplicación finalice

*Ilustración 35. Pseudocódigo para la planificación con sobresuscripción*

Este tipo de planificación pretende explotar el principio de que dos procesos ejecutando en un mismo núcleo de procesamiento, pueden ejecutar individualmente más lentamente, pero poseer un tiempo total de ejecución conjunta menor que una ejecución secuencial de los dos mismos procesos. Cabe

destacar, que las aplicaciones de comunicación no cumplen este principio, pues al sobresuscribir un proceso sobre un mismo núcleo con otro, el paso de mensajes entre procesos de la aplicación se torna más dificultosa.

Para demostrar este principio se han realizado pruebas con las distintas aplicaciones paralelas del sistema procurando que todas ellas tuviesen la misma duración individual de ejecución de 3600 segundos. La Tabla 93 muestra los tiempos de ejecución del sistema completo de una planificación secuencial con una carga de trabajo de dos aplicaciones en cada caso, procurando combinar el tipo de funcionalidad predominante, y la Tabla 94 presenta los tiempos obtenidos para las mismas cargas de trabajo con una planificación con sobresuscripción. Debido a que se produce el mismo resultado independiente del orden de sobresuscripción, se presentan los cálculos sin repetición.

| **Tipo de aplicación** | **Cpu** | **Memoria** |
|---|---:|---:|
| **Cpu** | 7200 s | 7200 s |
| **Memoria** | - | 7200 s |
| **Comunicación** | - | - |

*Tabla 93. Tiempo de ejecución de un sistema con planificación secuencial con dos aplicaciones*

| **Tipo de aplicación** | **Cpu** | **Memoria** |
|---|---:|---:|
| **Cpu** | 7140 s | 7130 s |
| **Memoria** | - | 7085 s |
| **Comunicación** | - | - |

*Tabla 94. Tiempo de ejecución de un sistema con planificación con sobresuscripción con dos aplicaciones*

Este principio se basa en que los ciclos de reloj de un núcleo de procesamiento en los que una aplicación está bloqueada a la espera de algún evento, pueden ser aprovechados para realizar la ejecución de otra aplicación. De esta manera, aunque el tiempo de ejecución individual de cada una de ellas aumenta debido a las penalizaciones por los cambios de contexto llevados a cabo, la ejecución completa de las dos aplicaciones disminuye al aprovechar los momentos de inactividad de las mismas.

De los tiempos obtenidos se puede calcular el *slowdown* o velocidad que se pierde de la planificación con sobresuscripción a la planificación secuencial, valores que muestra la Tabla 95.

| Tipo de aplicación | Cpu | Memoria |
|---|---|---|
| **Cpu** | 0.992 | 0.990 |
| **Memoria** | - | 0.984 |
| **Comunicación** | - | - |

*Tabla 95. Slowdown existente entre la planificación con sobresuscripción y la planificación secuencial*

Menores valores de *slowdown* indican mayor beneficio al sobresuscribir ese tipo de aplicaciones, debido a que aprovechan mejor sus tiempos de inactividad, de manera que cuando una aplicación está bloqueada la otra aplicación está ejecutando. La Ilustración 36 muestra el pseudocódigo del modelo de sobresuscripción desarrollado.

- Inicializar *slowdown* mínimo con un valor arbitrariamente alto
- Generar todas las combinaciones posibles de asignación de los procesos de la aplicación a los núcleos de procesamiento libres o con solo un proceso en ejecución
    - Analizar la combinación actual
        - Para cada proceso de la aplicación
            - Calcular el *slowdown* individual resultado de asignar el proceso actual al núcleo que indica la combinación de mapeo actual
            - Sumar *slowdown*
        - Si el *slowdown* total actual calculado es menor que el *slowdown* mínimo
            - Almacenar la combinación actual como combinación de mapeo a aplicar
            - Igualar el valor del *slowdown* mínimo al *slowdown* total actual

*Ilustración 36. Pseudocódigo del modelo de sobresuscripción*

El modelo utilizado por el algoritmo de sobresuscripción propuesto calcula el *slowdown* de cada combinación posible de mapeo de los procesos de una nueva aplicación a ejecutar con los procesos de las aplicaciones que se encuentran en ejecución en ese momento. De todas las combinaciones de mapeo posibles, escoge aquella que minimiza el *slowdown* del resultado de sobresuscribir dos procesos. Cabe destacar, que un núcleo de procesamiento totalmente libre supone un *slowdown* de valor nulo y que, por tanto, antes de aplicar sobresuscripción se buscará que cada proceso ejecute en un núcleo libre.

También es necesario mencionar, que el algoritmo recorre los procesos a ejecutar fuera del orden definido en la carga de trabajo.

Además, cada vez que una aplicación finaliza, se vuelve a calcular los posibles mapeos de las aplicaciones en ejecución, para comprobar si el cambio producido en el sistema beneficia una nueva configuración de mapeo de las aplicaciones.

Con el motivo de explicar detalladamente el funcionamiento del algoritmo de planificación con sobresuscripción se proponen una serie de escenarios que recojan distintas situaciones que se pueden establecer durante la planificación.

La Ilustración 37 presenta un escenario sencillo, con cuatro núcleos de procesamiento y tres aplicaciones, en el que no ocurre sobresuscripción de procesos.

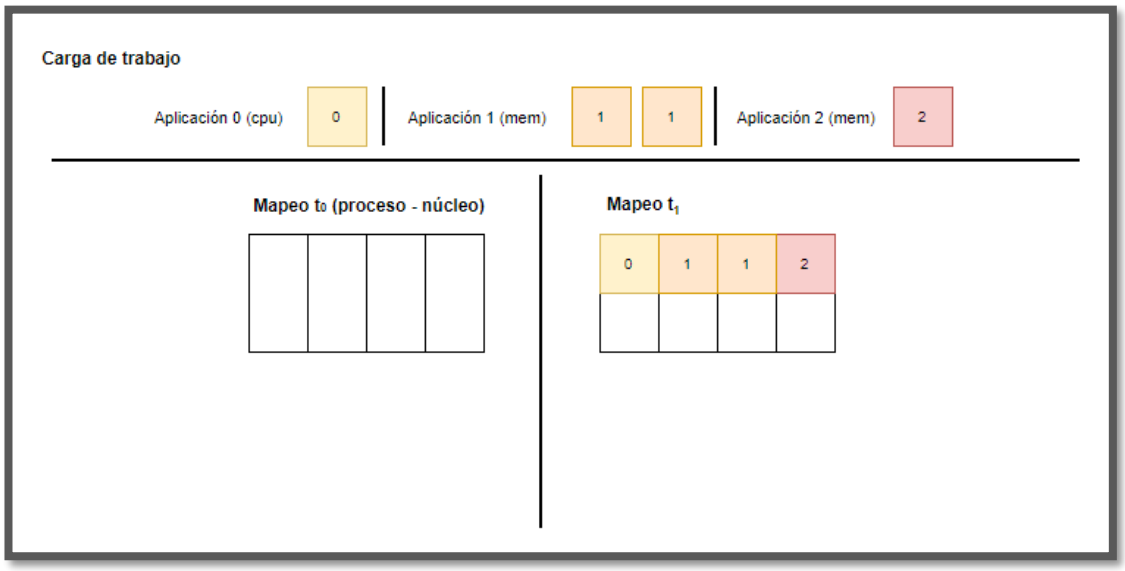



*Ilustración 37. Escenario 1 de planificación con sobresuscripción*

Como se puede observar, puesto que hay rescursos disponibles para todas las aplicaciones estas se mapean en núcleos libres del sistema, pues suponen la combinación de mapeo con el menor *slowdown* y por tanto la configuración de sobresuscripción óptima en este caso.

La Ilustración 38 presenta un escenario con sobresuscripción y una reconfiguración del mapeo.

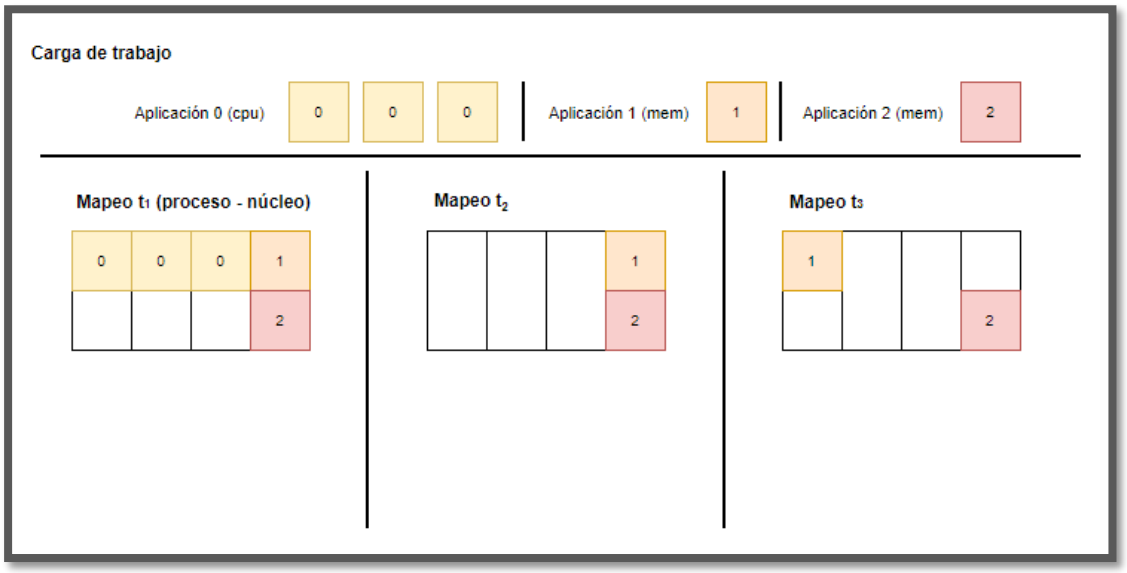



*Ilustración 38. Escenario 2 de planificación con sobresuscripción*

En $t_1$ se mapean las aplicaciones 0 y 1 a los núcleos de procesamiento libres. Posteriormente, se sobresuscribe la aplicación 2, intensiva en memoria, sobre la aplicación 1, intensiva en acceso a memoria también, puesto que en el cálculo de las posibles combinaciones de mapeo, es la que menor *slowdown* presenta.

En $t_2$ la aplicación 0 termina, por lo que quedan recursos libres, y la sobresuscripción de la aplicación 2 con la aplicación 1 no es la mejor configuración en este estado.

Por tanto, en $t_3$ se vuelven a realizar los cálculos de *slowdown* de las distintas combinaciones de mapeo y se selecciona la mínima, que consiste en volver a mapear el proceso de la aplicación 1 sobre un núcleo de procesamiento libre.

La Ilustración 39 muestra un escenario con más opciones de sobresuscripción.

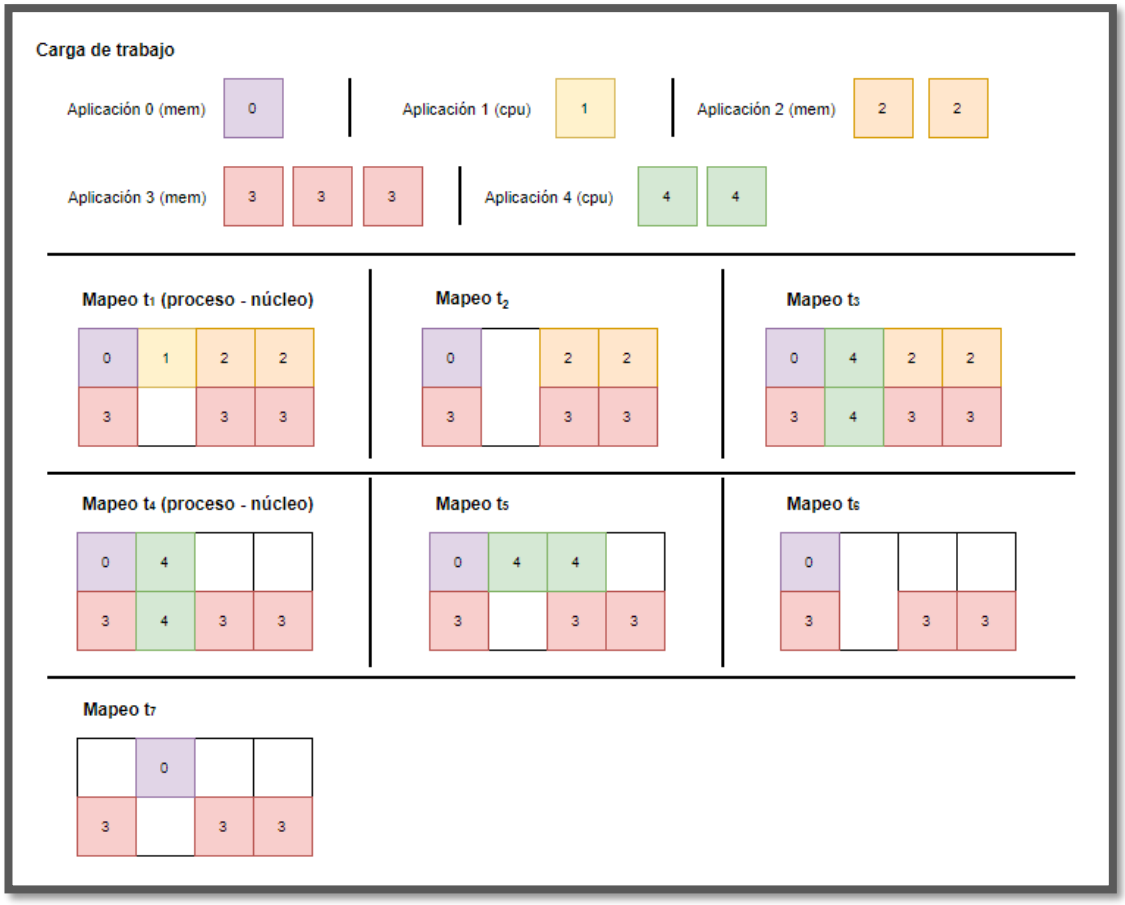



*Ilustración 39. Escenario 3 de planificación con sobresuscripción*

En el instante $t_1$ se planifican las aplicaciones 0, 1 y 2 en los núcleos de procesamiento libres del sistema. Porteriormente, se sobresuscribe la aplicación 3 que es intensiva en acceso a memoria, siendo la mejor combinación de mapeo aquella que involucra sobresuscribr un proceso con el proceso de la aplicación 0 tambíen de memoria y dos procesos con los procesos de la aplicación 2 intensiva en operaciones de memoria del mismo modo.

En $t_2$ termina la aplicación 1, y en $t_3$ se ejecuta la aplicación 4 intensiva en cálculos de procesamiento, cuya mejor combinación (la única disponible) de mapeo es asignar un proceso al núcleo que había quedado libre y sobresuscribirlo con su otro proceso.

En el instante $t_4$ finaliza su ejecución la aplicación 2, y se intenta volver a calcular las combinaciones posibles de mapeo de cada aplicación, empezando por la más antigua en la lista de aplicaciones en ejecución, para comprobar si hay una que implique un menor *slowdown*. De esta manera, en $t_5$ se determina que existe un mejor mapeo para la aplicación 4, que consiste en dejar un proceso en un núcleo de procesamiento libre y mapear el otro con un proceso intensivo en rutinas de comunicación.

Por último, en $t_6$ termina la aplicación 4 y queda un núcleo libre. Por tanto, el proceso de la aplicación 0 sobresuscrito con el proceso de la aplicación 3 se mapea sobre dicho núcleo, derivando en un estado sin sobresuscripción.

La Ilustración 40 presenta un escenario con aplicaciones de comunicación.

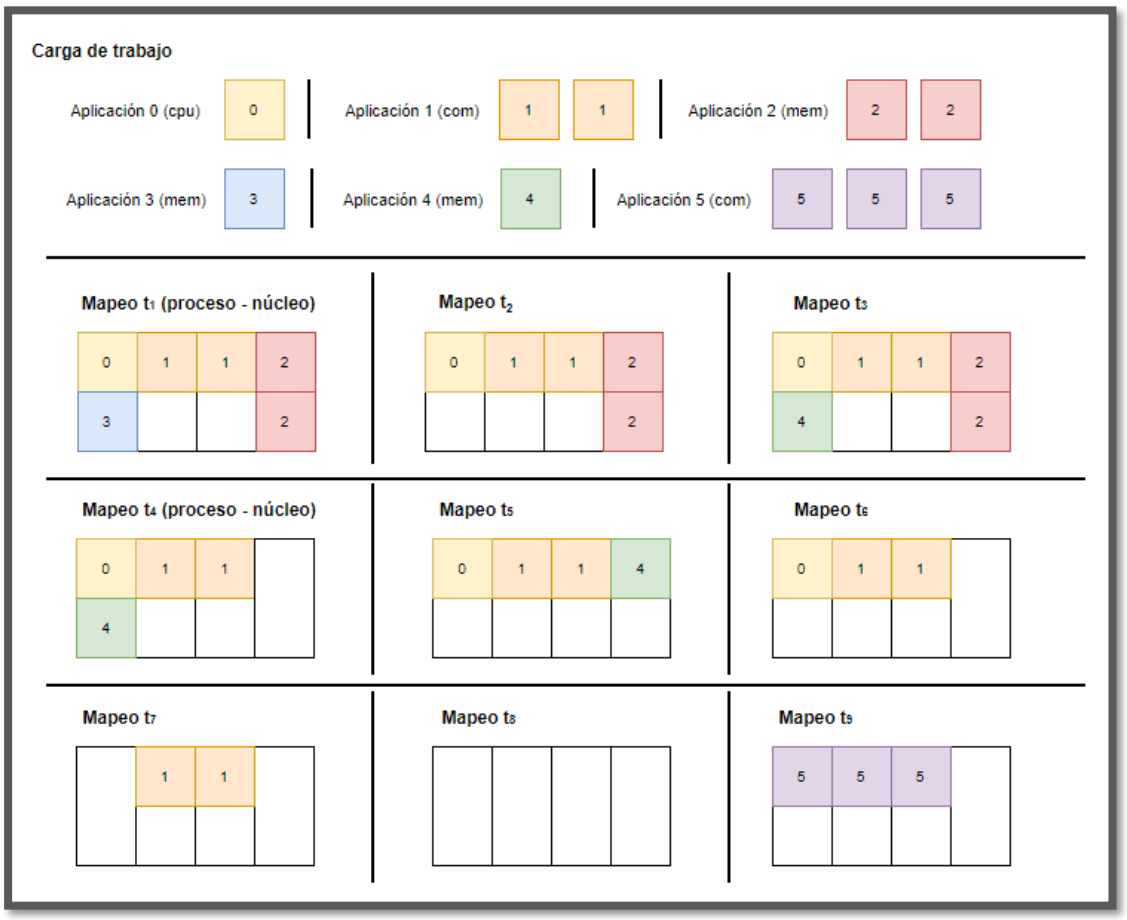



*Ilustración 40. Escenario 4 de planificación con sobresuscripción*

En el instante $t_1$, se asginan las aplicaciones 0 y 1 a los núcleos libres. La aplicación 2 asigna uno de los procesos al núcleo restante y su otro proceso lo sobresuscribe consigo mismo, pues su única alternativa es sobresuscribirse con un proceso intensivo en cpu que ofrece mayor *slowdown*, ya que las aplicaciones de comunicación no permiten la sobresuscripción de ningún tipo y necesitan un núcleo libre por proceso. La aplicación 3 se sobresuscribe con la 0, al no tener ninguna otra posibilidad. En $t_2$ finaliza la aplicación 3 y en $t_3$, la aplicación 4 se sobresuscribe con la 0. En $t_4$ finaliza la aplicación 2 y la aplicacción 4 reconfigura su mapeo asignándose el núcleo libre. La aplicación 5 no puede ejecutar debido a que al ser de comunicación no puede sobresuscribirse y por tanto tiene que esperar hasta el instante $t_8$ en el que hay suficientes núcleos libres para iniciarse.

### 4.4.3 Técnicas de caracterización

En este apartado se describen algunos de los modelos diseñados para la caracterización del comportamiento de aplicaciones. Cabe destacar, que en el presente proyecto, el comportamiento predominante de la aplicación solo es relevante para las planificaciones fuera de orden con maleabilidad y con sobresuscripción.

#### 4.4.3.1 Caracterización estática

Este tipo de caracterización se basa en la definición explícita del comportamiento de la aplicación. Es el propio usuario propietario de la aplicación quién determina si la aplicación es intensiva en procesamiento, en accesos a memoria o en el uso de rutinas de comunicación. Por tanto, no es necesario realizar ningún cálculo para la caracterización, si no que el tipo de aplicación viene dado en la carga de trabajo.

#### 4.4.3.2 Caracterización previa a la ejecución real

En este tipo de caracterización se realiza una falsa ejecución de las aplicaciones paralelas definidas en la carga de trabajo, de manera que durante periodo de tiempo la aplicación se ejecuta con el simple objetivo de determinar su comportamiento. Una vez la aplicación ha sido caracterizada, se interrumpe la ejecución, a través de un comando de control que se envía al módulo de comunicación de FlexMPI, y se pasa a caracterizar otra aplicación o se comienza la ejecución del sistema real si todas las aplicaciones ya han sido caracterizadas.

Para diferenciar las aplicaciones se hace uso de dos métricas monitorizadas en las mismas. La primera, consiste en un contador *hardware* que indica los accesos a memoria caché realizados y se establece un valor umbral de dicho contador. Si la aplicación analizada sobrepasa el umbral, se caracteriza como una aplicación intensiva en accesos de memoria, mientras que, si el valor monitorizado en la aplicación es inferior al umbral, se pasa a analizar la segunda métrica, que supone una medida recogida por el módulo de monitorización de FlexMPI y que indica el tiempo que se consume en la ejecución de rutinas de comunicación. Nuevamente se establece un valor umbral, si se sobrepasa la aplicación es caracterizada como intensiva en comunicación y en caso contrario se determina que es una aplicación predominante en cálculos de procesamiento.

#### 4.4.3.3 Caracterización dinámica

La caracterización dinámica se fundamenta en realizar el análisis de comportamiento de una aplicación durante la ejecución real de la misma.

Para ello, cuando el planificador con sobresuscripción inicia una nueva aplicación, la mapea sobre los núcleos de procesamiento sin obedecer a ninguna regla. Una vez iniciada, durante un periodo de tiempo definido en el sistema, se analiza el comportamiento de la aplicación y se caracteriza. Cuando la aplicación queda caracterizada, el planificador con sobresuscripción vuelve a mapear la aplicación, esta vez calculando la asignación de núcleos de procesamiento que minimiza el *slowdown* que determinan los distintos tipos de aplicaciones o bien es maleabilizada por el planificador con maleabilidad.

## 4.5 Planificación del proyecto

Este apartado indica la planificación temporal seguida durante el desarrollo del proyecto.

El proceso de desarrollo se compone de las fases de la metodología de trabajo descrita en el 4.1 Objetivos, metodología de trabajo y fases de desarrollo:

- **Fase de planificación y especificación**
  - Estudio de viabilidad, presupuesto y planificación temporal
  - Puesta a punto del entorno de desarrollo
  - Toma de requisitos
- **Fase de análisis**
  - Análisis de requisitos
  - Análisis de casos de uso
  - Elección de la solución y validación con el cliente
- **Fase de diseño**
  - Diseño del sistema
  - Validación con el cliente
- **Fase de codificación**
  - Implementación del sistema
- **Fase de verificación**
  - Plan de pruebas y trazabilidad
  - Verificación de diseño
  - Verficación de Análisis
  - Verificación de planificación y especificación
  - Evaluación de resultados
- **Fase de documentación**

Cabe destacar, que el tiempo consumido como parte de realizar ajustes que permitan una correcta verificación se incluye en dicha etapa de desarrollo.

La planificación inicial del proyecto se estimó en base a la cantidad de créditos que componen la asignatura de trabajo de fin de grado, la cual posee 12 créditos ECTS, que determinan una duración de trabajo total de 360 horas. Puesto que la convocatoria de entrega escogida resuelve la fecha de entrega la cuarta semana de junio de 2017, y el trabajo estimado por semana es de 18 horas, se calcula que el desarrollo comprenderá 20 semanas con su fecha de inicio en la última semana de enero.

Sin embargo, el desarrollo real no coincidió exactamente con la planificación inicial realizada. Los principales motivos de este hecho fueron una duración menor de la fase de planificación y especificación  a la duración estimada y problemas durante la fase de implementación que consumieron más tiempo del estimado.

Cabe destacar que durante las últimas semanas de desarrollo se puso énfasis en retocar los detalles de documentación que faltaban, empleando la totalidad del tiempo disponible.

La Ilustración 41 y la Ilustración 42 presenta los diagramas de Gantt de la planificación inicial estimada y la planificación real llevada a cabo.

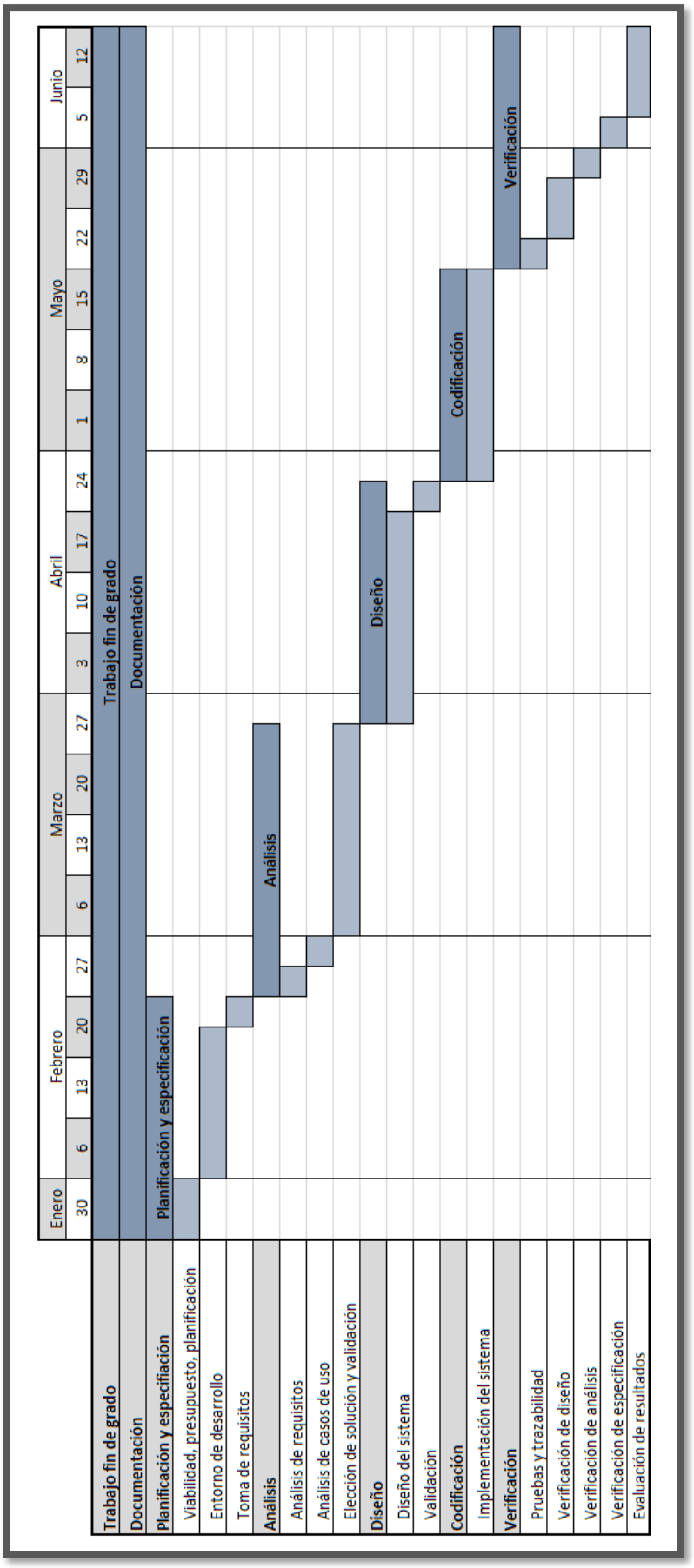



*Ilustración 41. Diagrama de Gantt de planificación inicial*

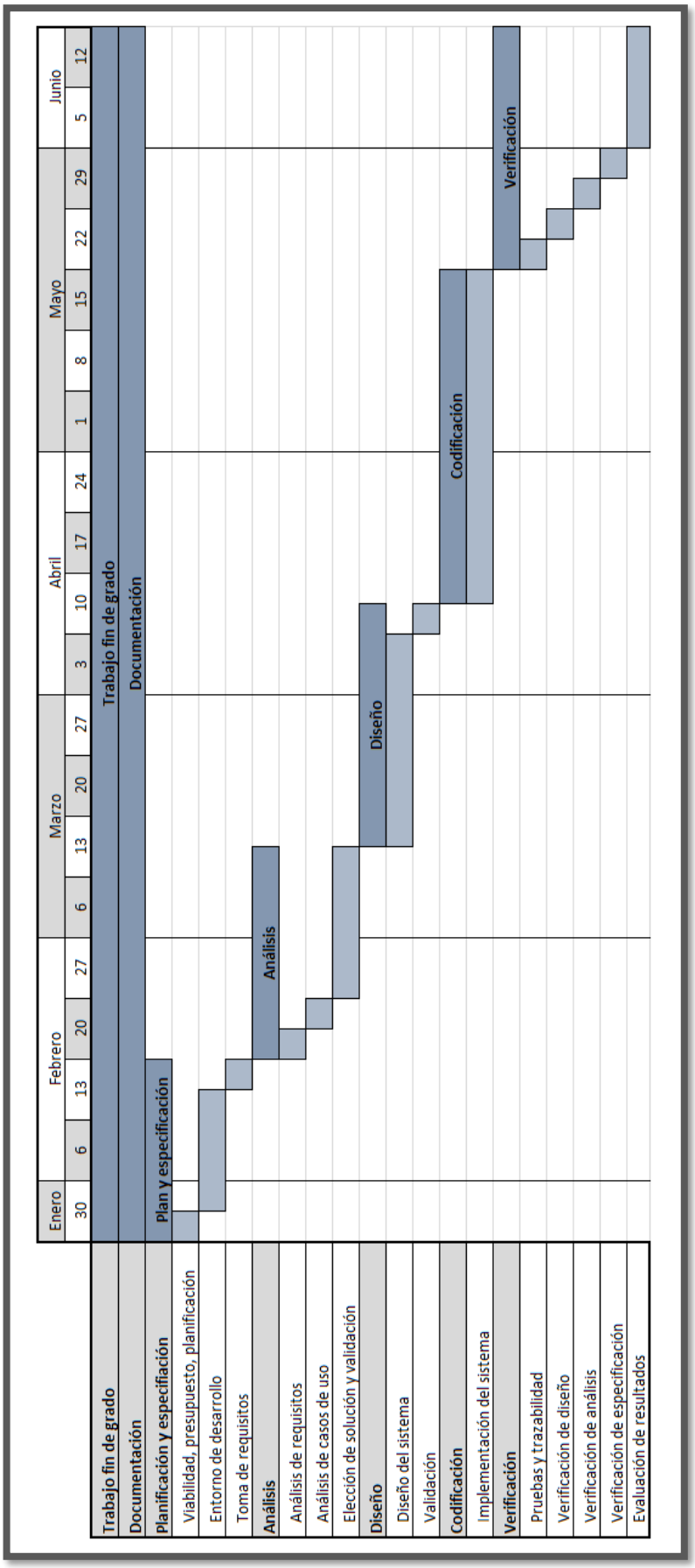



*Ilustración 42. Diagrama de Gantt de desarrollo real*

# Capítulo 5

# Evaluación

El presente capítulo evalúa el sistema teniendo en cuenta varias aproximaciones. En primer lugar, se presenta la plataforma *hardware* en la que se han llevado a cabo las diferentes pruebas. En segundo lugar, se presenta la especificación del plan de pruebas llevado a cabo junto con la matriz que permite trazar la relación entre los requisitos analizados del sistema y las pruebas diseñadas. Por último, se exponen diversas pruebas de rendimiento que permiten determinar el tiempo que consumen los diferentes métodos de planificación del sistema.

## 5.1 Descripción de la plataforma hardware

La plataforma sobre la que se han realizado las distintas pruebas del sistema es el *cluster Tucán* de la universidad Carlos III de Madrid [44], el cual supone un sistema de memoria distribuida. En concreto, se han utilizado tres nodos, cuyas características se presentan a continuación.

- Nodos compute-1-2 y compute-1-7. Estos nodos presentan exactamente las mismas arquitecturas. Cuentan con dos sockets con el procesador *Intel Xeon* E5405 que trabaja a una frecuencia de 2.00 GHz con 4 núcleos por procesador sin *hyperthreading*. Poseen dos niveles de caché el primero divido datos e instrucciones de 128 kilobytes y el segundo de 6 megabytes, y un tamaño de memoria principal de 8 gigabytes.
- Nodo compute-7-2. Cuenta con cuatro sockets que montan el procesador *Intel Xeon* E7- 4807 que posee una frecuencia de 1.87 GHz con 12 núcleos con tecnología *hyperthreading*. Posee tres niveles de memoria caché de 64 kilobytes, 256 kilobytes y 18 megabytes respectivamente y un tamaño de memoria principal de 128 gigabytes.

## 5.2 Especificación del plan de pruebas

Este apartado presenta el plan de pruebas diseñado en la fase de análisis conjuntamente con su resultado, con el objetivo de conservar la extensión de la presente documentación.

Cabe destacar, que se revisarán tres tipos de pruebas verificadoras, que suponen las pruebas de codificación o implantación, de diseño, de validación de la funcionalidad y de cumplimiento de requisitos o especificaciones.

La información de cada prueba se presentará en forma tabular. muestra la plantilla que seguirán los requisitos expuestos.

| | |
|---|---|
| **ID** | P-XX |
| **Objetivo** | Objetivo de la prueba |
| **Precondiciones** | Condiciones anteriores a la prueba |
| **Secuencia** | Descripción de la ejecución |
| **Postcondiciones** | Condiciones posteriores a la prueba |
| **Resultado** | Verificado o No verificado |
| **Trazabilidad** | Requisitos analizados |

*Tabla 96. Plantilla tabular para la presentación de pruebas*

Los atributos que definen una prueba del sistema son los siguientes:

- **ID:** Código que identifica de forma unívoca a cada prueba (P significa prueba y XX un número de dos dígitos).
- **Objetivo:** Objetivo perseguido al efectuar la prueba.
- **Precondiciones:** Condiciones que debe cumplir el sistema antes de efectuar la prueba.
- **Secuencia:** Pasos que se deben efectuar para realizar la prueba.
- **Postcondiciones:** Condiciones que se deben cumplir tras efectuar la prueba.
- **Resultado:** Resultado obtenido tras la ejecución de la prueba. Puede ser verificado o no verificado.
- **Trazabilidad:** Relación con los requisitos del sistema analizados.

A continuación, se presentan las pruebas diseñadas para el presente proyecto y su resultado.

| | |
|---|---|
| **ID** | P-01 |
| **Objetivo** | Verificar que el sistema solo ejecuta las aplicaciones indicadas en la carga de trabajo |
| **Precondiciones** | Carga de trabajo debe contener definición de aplicaciones |
| **Secuencia** | Rellenar la carga de trabajo con el formato pertinente y arrancar el sistema con cada una de las planificaciones |
| **Postcondiciones** | El sistema ejecuta las aplicaciones indicadas en la carga de trabajo con el número de procesos y tipo especificados |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-01, RF-12, RNF-01, RNF-02, RNF-03, RNF-04, RNF-05 |

*Tabla 97. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-02 |
| **Objetivo** | Verificar que el controlador recibe instrucciones por entrada estándar |
| **Precondiciones** | El controlador debe haber iniciado su ejecución |
| **Secuencia** | Ejecutar el controlador e introducir datos por entrada estándar |
| **Postcondiciones** | El sistema recibe continuamente los datos introducidos |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-02 |

*Tabla 98. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-03 |
| **Objetivo** | Verificar que el controlador selecciona la política de planificación por entrada estándar |
| **Precondiciones** | El controlador debe haber iniciado su ejecución |
| **Secuencia** | Ejecutar el controlador e introducir las instrucciones de selección de tipo de planificación |
| **Postcondiciones** | El sistema reconoce las instrucciones y selecciona la política de planificación indicada |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-03 |

*Tabla 99. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-04 |
| **Objetivo** | Verificar que el controlador planifica el orden de ejecución de las aplicaciones conforme a la lógica de la política escogida |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y haber seleccionado la política deseada |
| **Secuencia** | Ejecutar el controlador e introducir las instrucciones de selección de cada una de las políticas de planificación |
| **Postcondiciones** | El sistema ejecuta las aplicaciones conforme a la lógica de la política de planificación |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-04, RF-05, RF-06, RF-07, RF-14, RNF-05 |

*Tabla 100. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-05 |
| **Objetivo** | Verificar que el controlador arranca la planificación por indicación de entrada estándar |
| **Precondiciones** | El controlador debe haber iniciado su ejecución |
| **Secuencia** | Ejecutar el controlador e introducir la instrucción que indica el arranque de la planificación de aplicaciones |
| **Postcondiciones** | El sistema comienza la planificación de aplicaciones |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-08 |

*Tabla 101. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-06 |
| **Objetivo** | Verificar que el controlador aplica por defecto la planificación secuencial |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y no seleccionar ninguna política de planificación |
| **Secuencia** | Ejecutar el controlador e introducir la instrucción que indica el arranque de la planificación de aplicaciones sin indicar ninguna política de planificación |
| **Postcondiciones** | El sistema comienza aplicando planificación secuencial |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-04, RF-09 |

*Tabla 102. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-07 |
| **Objetivo** | Verificar que el controlador es capaz de realizar una suspensión de la planificación |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y la planificación debe estar en marcha |
| **Secuencia** | Introducir por entrada estándar la instrucción que indica la suspensión de la planificación |
| **Postcondiciones** | El sistema detiene la planificación |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-10 |

*Tabla 103. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-08 |
| **Objetivo** | Verificar que el controlador es capaz de reanudar el sistema de planificación |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y la planificación debe estar suspendida |
| **Secuencia** | Introducir por entrada estándar la instrucción que indica la reanudación de la planificación |
| **Postcondiciones** | El sistema reanuda la planificación |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-11 |

*Tabla 104. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-09 |
| **Objetivo** | Verificar que el controlador determina el estado de los núcleos de procesamiento del sistema para iniciar aplicaciones y finalizarlas |
| **Precondiciones** | El controlador debe haber iniciado su ejecución |
| **Secuencia** | Ejecutar el controlador y arrancar la planificación |
| **Postcondiciones** | Ante el inicio y finalización de aplicaciones, el controlador determina correctamente los núcleos de procesamiento libres y ocupados |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-13, RF-14 |

*Tabla 105. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-10 |
| **Objetivo** | Verificar que el controlador se comunica remotamente con las aplicaciones para activar la monitorización |
| **Precondiciones** | El controlador debe haber iniciado su ejecución |
| **Secuencia** | Ejecutar el controlador y arrancar la planificación |
| **Postcondiciones** | Ante el inicio de una aplicación el controlador crea el hilo que se encarga de indicar el inicio de monitorización y envía dicha instrucción a la aplicación iniciada |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-15 |

*Tabla 106. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-11 |
| **Objetivo** | Verificar las aplicaciones se comunican remotamente con el controlador para enviar los datos de monitorización |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y comenzar la planificación, iniciando alguna aplicación a la que se envía la activación de la monitorización |
| **Secuencia** | Ejecutar el controlador y arrancar la planificación |
| **Postcondiciones** | La aplicación comienza a enviar sus datos de monitorización pertinentes. |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-16, RNF-06 |

*Tabla 107. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-12 |
| **Objetivo** | Verificar que el controlador almacena los diez resultados de monitorización más recientes |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y comenzar la planificación, iniciando alguna aplicación que se encuentre en monitorización |
| **Secuencia** | Ante cada resultado recibido comprobar la estructura de almacenamiento |
| **Postcondiciones** | La estructura de almacenamiento de datos de monitorización solo contiene los 10 resultados recibidos más recientes, descartando los antiguos |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-17 |

*Tabla 108. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-13 |
| **Objetivo** | Verificar que el controlador recibe la indicación de finalización por parte de las aplicaciones |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y comenzar la planificación, iniciando alguna aplicación que se encuentre en monitorización |
| **Secuencia** | Arrancar la planificación y esperar la terminación de alguna aplicación |
| **Postcondiciones** | Ante la terminación de las aplicaciones el controlador recibe la notificación y actualiza los recursos disponibles. |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-18 |

*Tabla 109. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-14 |
| **Objetivo** | Verificar que el controlador aplica maleabilidad atendiendo a la lógica diseñada |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y debe haber seleccionado la planificación fuera de orden con maleabilidad |
| **Secuencia** | Seleccionar la planificación fuera de orden con maleabilidad y analizar una situación en la que se pueda aplicar maleabilidad |
| **Postcondiciones** | Ante la aparición de la posibilidad de aplicación de maleabilidad, el controlador incrementa el número de procesos de las aplicaciones conforme a la lógica descrita |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-19 |

*Tabla 110. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-15 |
| **Objetivo** | Verificar que el controlador comunica remotamente a las aplicaciones el incremento que deben realizar de su número de procesos |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y debe haber seleccionado la planificación fuera de orden con maleabilidad |
| **Secuencia** | Seleccionar la planificación fuera de orden con maleabilidad y analizar una situación en la que se pueda aplicar maleabilidad |
| **Postcondiciones** | Ante la aparición de la posibilidad de aplicación de maleabilidad, el controlador incrementa el número de procesos de las aplicaciones conforme a la lógica descrita y envía a las aplicaciones involucradas la instrucción de incremento del número de procesos |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-20 |

*Tabla 111. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-16 |
| **Objetivo** | Verificar que el controlador comunica remotamente a las aplicaciones los núcleos en los que deben mapearse cada uno de sus procesos |
| **Precondiciones** | El controlador debe haber iniciado su ejecución y arrancado la planificación |
| **Secuencia** | Ejecutar el controlador y arrancar la planificación |
| **Postcondiciones** | Ante un inicio de aplicación, aplicación de maleabilidad o mapeo en busca de una mejor configuración de mapeo, el controlador envía a las aplicaciones la instrucción de mapeo procesos-núcleos |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-21 |

*Tabla 112. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-17 |
| **Objetivo** | Verificar que el controlador se comunica remotamente con las aplicaciones para forzar su terminación ante un error del sistema |
| **Precondiciones** | El controlador debe haber iniciado su ejecución, arrancado la planificación y determinar un error |
| **Secuencia** | Ejecutar el controlador, arrancar la planificación y forzar los errores comprobados |
| **Postcondiciones** | Ante la aparición del error, el controlador envía a las aplicaciones en ejecución la instrucción de terminación |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-22, RF-23 |

*Tabla 113. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-18 |
| **Objetivo** | Verificar que el controlador muestra los mensajes pertinentes por salida estándar |
| **Precondiciones** | El controlador debe haber iniciado su ejecución. |
| **Secuencia** | Ejecutar el controlador, arrancar la planificación. Posteriormente, ejecutar el controlador y forzar los errores comprobados |
| **Postcondiciones** | El controlador muestra los mensajes de cada una de las siguientes situaciones:<br>– Información de error<br>– Información de inicialización<br>– Información de entrada de datos<br>– Información de la suspensión<br>– Información de la reanudación<br>– Información de inicio de aplicación<br>– Información de la finalización de aplicación<br>– Información de la maleabilidad<br>– Información de mapeo procesos-núcleos<br>– Información de finalización del sistema |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-24, RF-25, RF-26, RF-27, RF-28, RF-29, RF-30, RF-31, RF-32, RF-33 |

*Tabla 114. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-19 |
| **Objetivo** | Verificar que el controlador permite la finalización del sistema por entrada estándar |
| **Precondiciones** | El controlador debe haber iniciado su ejecución |
| **Secuencia** | Introducir por entrada estándar la instrucción de finalización del sistema |
| **Postcondiciones** | El sistema finaliza su ejecución |
| **Resultado** | Verificado |
| **Trazabilidad** | RF-34 |

*Tabla 115. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-20 |
| **Objetivo** | Verificar que las aplicaciones paralelas despliegan y ejecutan correctamente la biblioteca externa PAPI |
| **Precondiciones** | Integración de la biblioteca externa PAPI |
| **Secuencia** | Ejecución de aplicación paralela y comienzo de la monitorización |
| **Postcondiciones** | Los datos son monitorizados correctamente |
| **Resultado** | Verificado |
| **Trazabilidad** | RNF-07 |

*Tabla 116. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-21 |
| **Objetivo** | Verificar que las aplicaciones paralelas despliegan y ejecutan correctamente la herramienta FlexMPI |
| **Precondiciones** | Integración de la biblioteca externa FlexMPI |
| **Secuencia** | Ejecución de aplicación paralela y ejecución de instrucciones de inicio de monitorización, de mapeo, de terminación y de maleabilidad |
| **Postcondiciones** | La aplicación ejecuta correctamente y atiende a las instrucciones que indican las funcionalidades pertinentes |
| **Resultado** | Verificado |
| **Trazabilidad** | RNF-08 |

*Tabla 117. Prueba del sistema P-01*

| | |
|---|---|
| **ID** | P-22 |
| **Objetivo** | Verificar que el sistema es compatible con una arquitectura de memoria distribuida |
| **Precondiciones** | Despliegue del sistema en una arquitectura de memoria distribuida |
| **Secuencia** | Ejecución del sistema |
| **Postcondiciones** | El sistema ejecuta correctamente |
| **Resultado** | Verificado |
| **Trazabilidad** | RNF-09 |

*Tabla 118. Prueba del sistema P-01*

## 5.3 Matriz de trazabilidad

Con el objetivo de presentar la relación existente entre los requisitos del sistema y las pruebas diseñadas y llevadas a cabo se expone la matriz de trazabilidad del presente proyecto en la Ilustración 43. Se verifican 43 requisitos, 34 funcionales y 9 no funcionales, a través de 22 pruebas de validación.

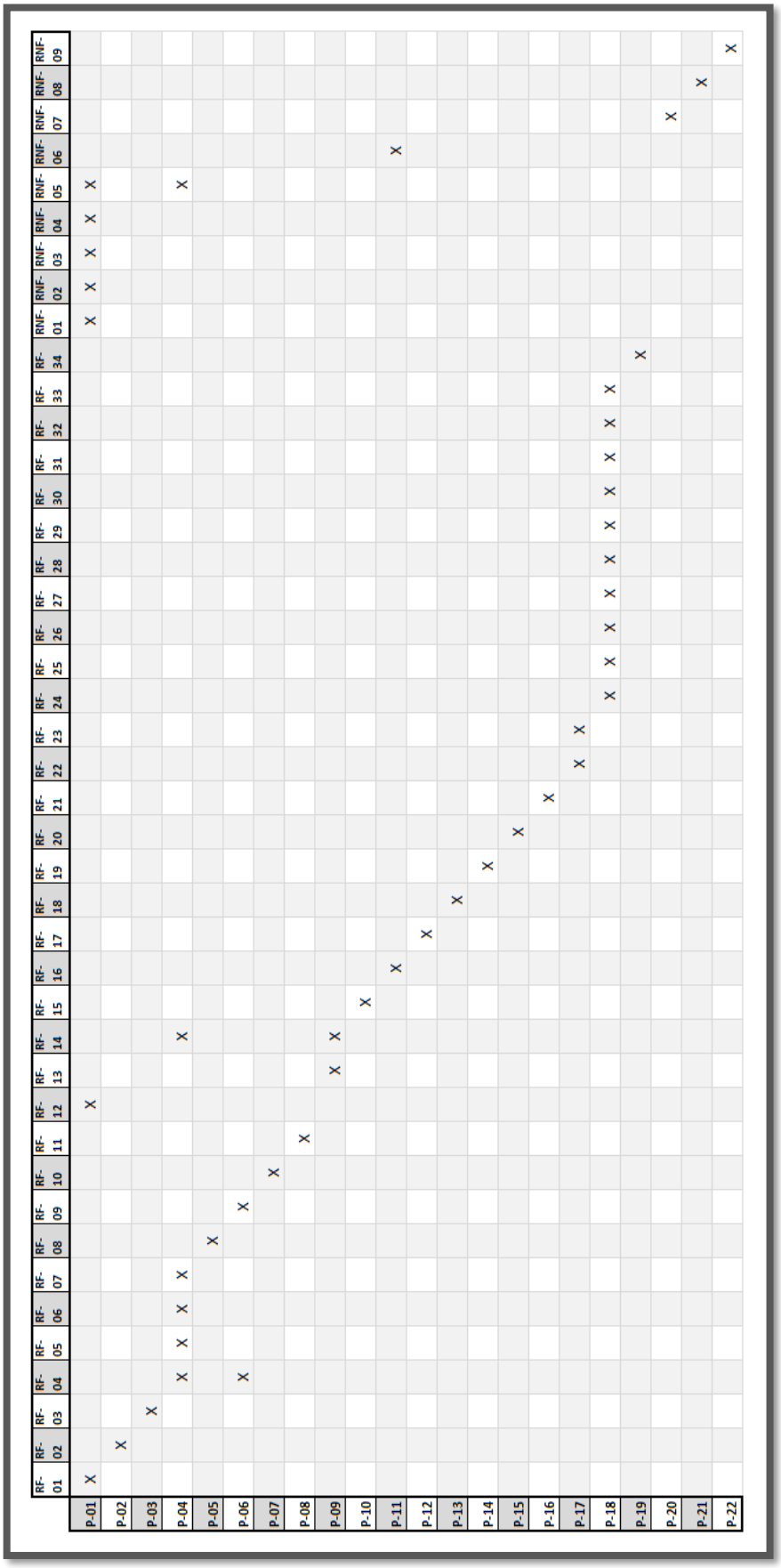



*Ilustración 43. Matriz de trazabilidad*

## 5.4 Pruebas de rendimiento

En este apartado se presentan diferentes pruebas de rendimiento que permitan evaluar cada una de las técnicas de planificación desarrolladas. Para ello se proponen cuatro escenarios diseñados y un escenario con un conjunto aleatorio de aplicaciones. Los escenarios diseñados se establecerán intentando exponer las bondades de cada política de planificación para poder analizar en qué casos podría resultar mejor aplicar cada una de ellas.

Cabe destacar que, por simplicidad, las pruebas utilizarán únicamente cuatro núcleos de procesamiento de cada nodo, asegurando que las aplicaciones dispongan su contexto en un mismo *socket* del nodo en el que se desarrollan las pruebas. Este hecho no restará generalidad a los resultados obtenidos.

### 5.4.1 Primer escenario diseñado

Este primer escenario pretende presentar aquellos casos en los que puede resultar eficiente aplicar una planificación secuencial al sistema. Esta política de planificación pretende dar prioridad al orden de llegada de las aplicaciones al sistema para su ejecución. Sin embargo, debido a que se basa en asegurar todos los recursos que necesitan las aplicaciones individualmente, en el orden en el que llegaron al sistema, los casos en los que la planificación secuencial resulta beneficiosa son escasos. La Tabla 119 muestra la carga de trabajo definida para este escenario.

| Tipo de aplicación | Número de procesos |
|---|---|
| Procesamiento | 4 |
| Comunicación | 4 |
| Memoria | 4 |
| Memoria | 2 |
| Memoria | 2 |
| Comunicación | 4 |
| Procesamiento | 4 |

*Tabla 119. Carga de trabajo del escenario de pruebas 1*

Como se puede observar, planificar secuencialmente las aplicaciones definidas, asegura que se utilizan los cuatro núcleos de procesamiento del sistema en todo momento, sin tener que realizar esperas que desaprovechen recursos. La Tabla 120 y la Ilustración 44 presentan los resultados obtenidos.

| Tiempo secuencial (s) | Tiempo fuera de orden (s) | Tiempo con maleabilidad (s) | Tiempo con sobresuscripción (s) |
|---|---|---|---|
| 975.05 | 980.06 | 980.04 | 972.08 |

*Tabla 120. Tiempos de planificación para el escenario 1*

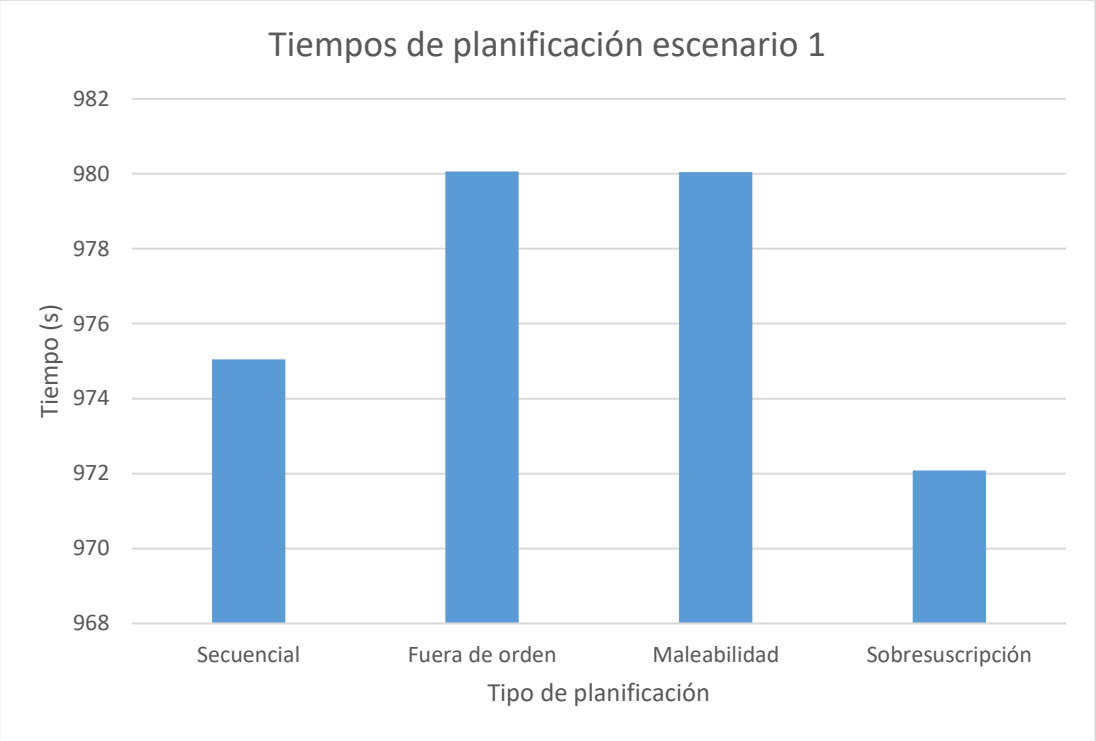



*Ilustración 44. Tiempos de planificación para el escenario 1*

A pesar de que este sea un caso realmente beneficioso para la planificación secuencial, se puede observar que los tiempos que obtienen las planificaciones fuera de orden y fuera de orden con maleabilidad, se encuentran en el mismo orden debido a que las aplicaciones de la carga de trabajo están dispuestas de manera que se ejecutan en orden, sin desaprovechar núcleos de procesamiento. Las variaciones se deben a la variabilidad que presentan las aplicaciones de comunicación en su ejecución en función de si las rutinas de comunicación entre procesos ejecutan más rápidamente.

Por otra parte, la planificación con sobresuscripción no obtiene demasiado beneficio, ya que no se han dado las mejores combinaciones de asignaciones con las aplicaciones definidas, y el margen de mejora se puede haber debido a la variabilidad comentada.

### 5.4.2 Segundo escenario diseñado

El presente escenario busca el objetivo de resaltar aquellos casos en los que la planificación fuera del orden definido en la carga de trabajo puede resultar en un beneficio muy amplio frente a una planificación secuencial. Dichos casos se componen de una carga de trabajo en la que aplicaciones de numerosos procesos bloquean la ejecución de otras con menos procesos que podrían ejecutar en núcleos de procesamiento libres. La Tabla 121 muestra la carga de trabajo definida para este escenario.

| Tipo de aplicación | Número de procesos |
|---|---|
| Procesamiento | 1 |
| Comunicación | 4 |
| Memoria | 1 |
| Comunicación | 4 |
| Memoria | 1 |
| Comunicación | 4 |
| Procesamiento | 1 |
| Comunicación | 4 |

*Tabla 121. Carga de trabajo del escenario de pruebas 2*

Como se puede observar, se intercalan aplicaciones de un solo proceso con aplicaciones de cuatro procesos. Este escenario provocará en la planificación secuencial que las aplicaciones de cuatro procesos tengan que esperar a que terminen las aplicaciones de un solo proceso mientras se desaprovechan tres núcleos de procesamiento. Además, la configuración dispuesta hace que no se pueda aplicar maleabilidad, pues fuera de orden siempre se aprovecharán los cuatro núcleos de procesamiento, y que no se pueda aplicar sobresuscripción, pues las aplicaciones de comunicación no lo admiten.

La Tabla 122 y la Ilustración 45 presentan los resultados obtenidos.

| Tiempo secuencial (s) | Tiempo fuera de orden (s) | Tiempo con maleabilidad (s) | Tiempo con sobresuscripción (s) |
|---|---|---|---|
| 2380,08 | 1470.05 | 1465.16 | 1480.05 |

*Tabla 122. Tiempos de planificación para el escenario 2*

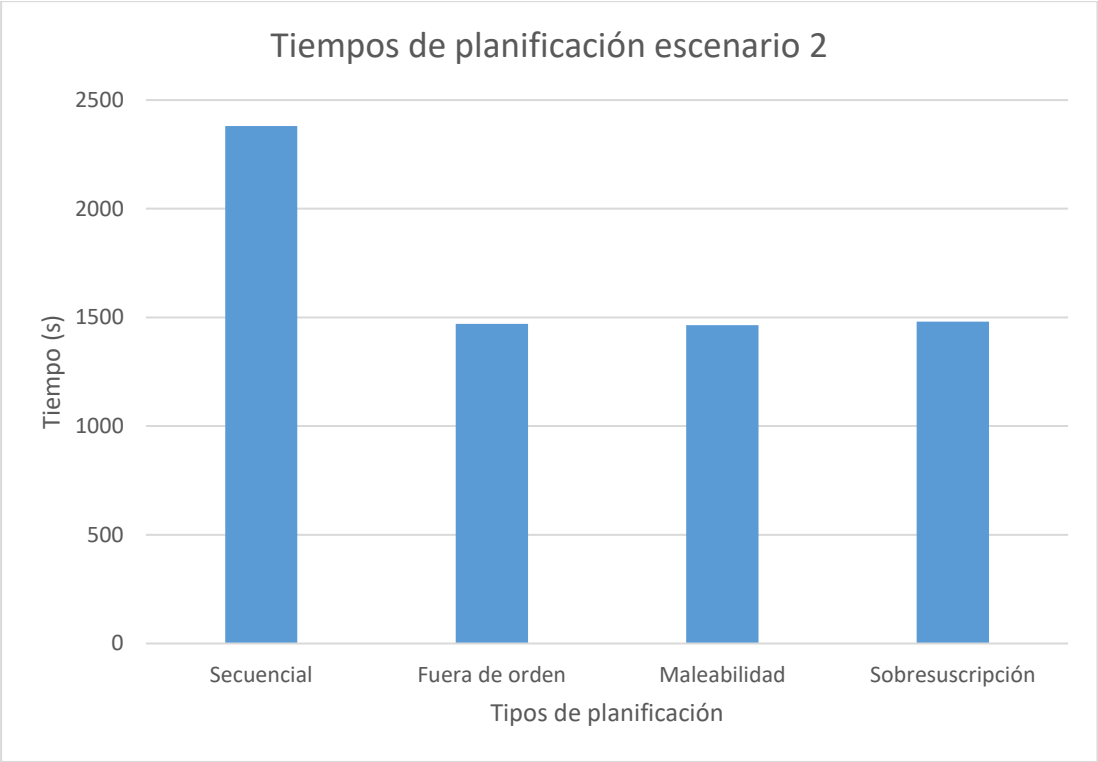



*Ilustración 45. Tiempos de planificación para el escenario 2*

Los resultados obtenidos reflejan claramente el beneficio de ejecutar las aplicaciones fuera de orden, aplicada por las políticas de planificación de fuera de orden, con maleabilidad y con sobresuscripción. En este caso se aprovechan todos los recursos del sistema en cada momento, pues las aplicaciones que cuentan con tan solo un proceso se ejecutan de forma paralela, mientras que la planificación secuencial, que persigue el objetivo de otorgar la ejecución en el orden expuesto, debe esperar la terminación de dichas aplicaciones dejando tres núcleos de procesamiento desocupados.

Entre los tres planificadores que aplican ejecución fuera de orden, no existen demasiadas diferencias debido a que la situación presentada no permite aplicar maleabilidad, pues en ningún momento quedan núcleos de procesamiento libres, ni sobresuscripción, pues las aplicaciones de comunicación no lo permiten.

### 5.4.3 Tercer escenario diseñado

Este escenario pretende mostrar los beneficios asociados a una política de planificación fuera de orden con maleabilidad. Estos beneficios se manifiestan cuando hay núcleos de procesamiento libres en el sistema que no son suficientes para poder ejecutar otras aplicaciones que requieren como mínimo un número

de procesos superior al de los recursos libres. Por tanto, ante esta situación, en lugar de dejar el núcleo desaprovechado, se determina un incremento en los procesos de las aplicaciones en ejecución en ese momento, ocupando todos los núcleos libres del sistema. La Tabla 123 muestra la carga de trabajo definida para este escenario.

| Tipo de aplicación | Número de procesos |
|---|---|
| Procesamiento | 3 |
| Comunicación | 4 |
| Memoria | 2 |
| Comunicación | 4 |
| Procesamiento | 3 |
| Comunicación | 4 |
| Memoria | 3 |

*Tabla 123. Carga de trabajo del escenario de pruebas 3*

Como se puede observar, la carga de trabajo definida no permite ejecutar aplicaciones fuera de orden. Sin embargo, hay aplicaciones que no aprovechan todos los recursos del sistema mientras se ejecutan. Con esta situación se espera que dichas aplicaciones aumenten el número de procesos ejecutando en paralelo, y que, por tanto, se reduzcan los tiempos frente a las planificaciones que desaprovechan recursos durante estos instantes.

La Tabla 124 y la Ilustración 46 muestran la carga de trabajo definida para este escenario.

| Tiempo secuencial (s) | Tiempo fuera de orden (s) | Tiempo con maleabilidad (s) | Tiempo con sobresuscripción (s) |
|---|---|---|---|
| 1550.06 | 1550.06 | 1400.04 | 1570.05 |

*Tabla 124. Tiempos de planificación para el escenario 3*

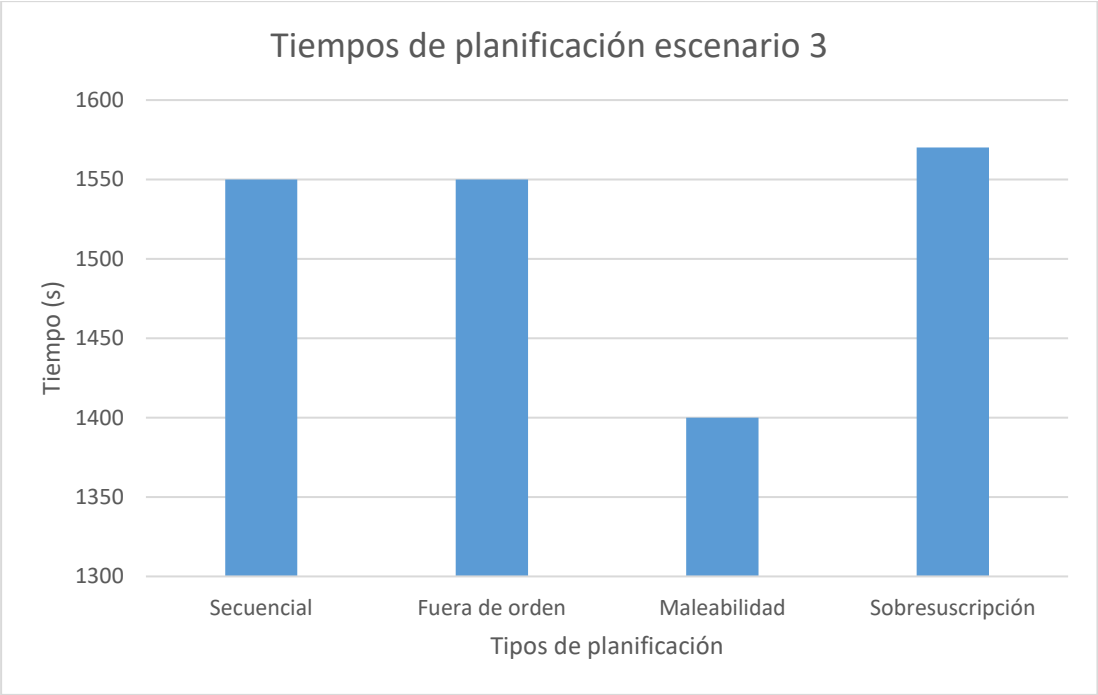



*Ilustración 46. Tiempos de planificación para el escenario 3*

Los resultados obtenidos muestran el claro beneficio que se obtiene al aplicar la maleabilidad de procesos en este caso. Al incrementar el paralelismo de las aplicaciones en un instante en el que ninguna otra aplicación se puede ejecutar, se permite la terminación de las aplicaciones, y, por tanto, del sistema completo, en un tiempo menor. Además, cabe destacar, que la última aplicación del sistema, que en otros tipos de planificación desaprovecha recursos de una forma clara, en la política que aplica maleabilidad, utiliza todos los recursos disponibles.

Por otra parte, los planificadores, secuencial y fuera de orden, se comportan de la misma manera debido a que no existe ninguna aplicación con un número pequeño de procesos que pueda ejecutarse en los núcleos que quedan libres en ningún instante. La planificación con sobresuscripción no puede manifestar sus beneficios pues no asigna una combinación óptima de la compartición de los núcleos de procesamiento, y, por tanto, su comportamiento también es semejante a los planificadores anteriores.

### 5.4.4 Cuarto escenario diseñado

Este último escenario escogido tiene el objetivo de mostrar los beneficios que pueden obtenerse de aplicar la planificación con sobresuscripción. Los casos más favorables suponen aquellos en los que en el sistema hay aplicaciones con un

comportamiento intensivo en memoria, ya que la compartición de un núcleo de procesamiento entre este tipo de aplicaciones permite su mayor aprovechamiento. La Tabla 125 muestra la carga de trabajo definida para este escenario.

| Tipo de aplicación | Número de procesos |
|---|---|
| Memoria | 4 |
| Memoria | 4 |
| Memoria | 4 |
| Memoria | 4 |
| Memoria | 4 |
| Memoria | 4 |
| Memoria | 4 |
| Memoria | 4 |

*Tabla 125. Carga de trabajo del escenario de pruebas 4*

La carga de trabajo no permite aplicar maleabilidad ni ejecución fuera de orden. Además, todas las aplicaciones son intensivas en memoria para acentuar el efecto beneficioso de la sobresuscripción en estos casos.

La Tabla 126 y la Ilustración 47 presentan los resultados obtenidos.

| Tiempo secuencial (s) | Tiempo fuera de orden (s) | Tiempo con maleabilidad (s) | Tiempo con sobresuscripción (s) |
|---|---|---|---|
| 960.05 | 960.04 | 960.06 | 944.10 |

*Tabla 126. Tiempos de planificación para el escenario 4*

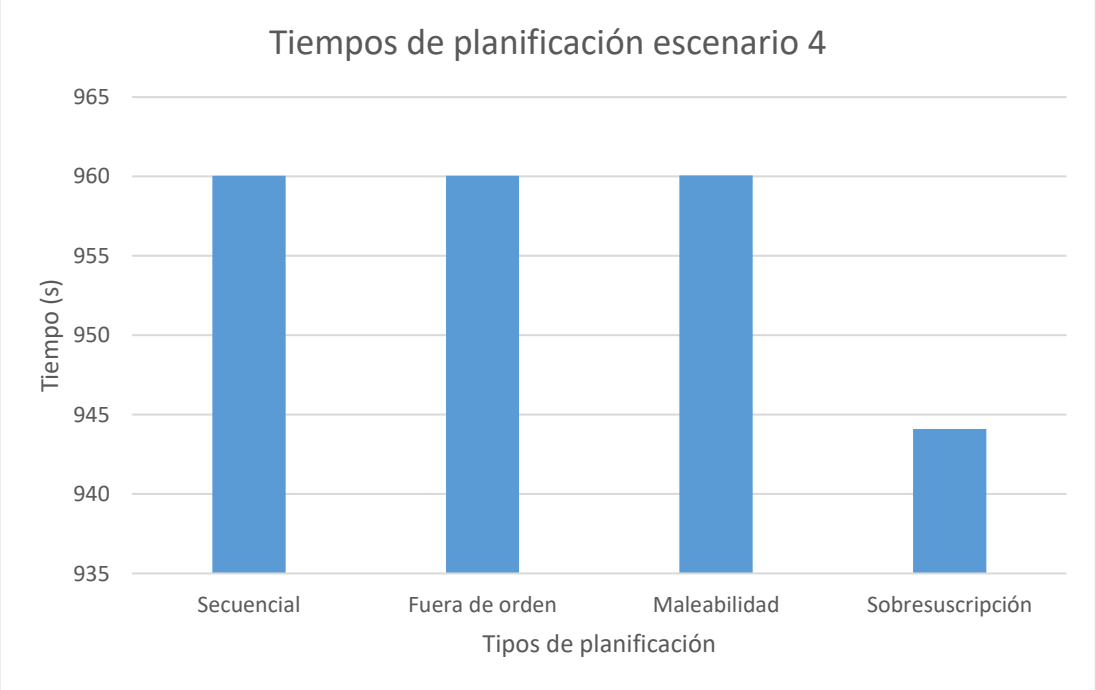



*Ilustración 47. Tiempos de planificación para el escenario 4*

Los resultados obtenidos indican que en efecto se produce un aprovechamiento mayor del núcleo de procesamiento. Además, las aplicaciones de memoria resultan buenas en este tipo de planificación, debido a que cuentan con numerosos ciclos de inactividad en el acceso a memoria, que se pueden solapar con los ciclos en la que la otra aplicación debe realizar cálculos y utilizar el núcleo de procesamiento. Sin embargo, este escenario supone un caso extremo, y siempre se cuenta con diversos factores que pueden derivar en que la sobresuscripción produzca tiempos de ejecución mayores al del sistema de aplicaciones completo secuencialmente, como por ejemplo que diversas aplicaciones no sean compatibles para la sobresuscripción al estar inactivas en los mismos instantes de tiempo.

Por otra parte, las demás políticas de planificación se comportan de una manera similar en este escenario, por no poder aplicar ejecuciones fuera de orden ni maleabilidad de procesos.

### 5.4.5 Escenario aleatorio

Este escenario pretende simular una carga de trabajo de real en la que la disposición de las aplicaciones no ha sido escogida, sino que la llegada de diferentes aplicaciones en la carga de trabajo es aleatoria. La Tabla 127 muestra la carga de trabajo elegida mediante un proceso aleatorio.

| Tipo de aplicación | Número de aplicaciones | Número de procesos |
|---|---|---|
| Procesamiento | 2 | 1 |
| Procesamiento | 7 | 2 |
| Procesamiento | 1 | 3 |
| Procesamiento | 1 | 4 |
| Memoria | 2 | 1 |
| Memoria | 2 | 2 |
| Memoria | 4 | 3 |
| Memoria | 2 | 4 |
| Comunicación | 1 | 1 |
| Comunicación | 1 | 2 |
| Comunicación | 2 | 3 |
| Comunicación | 2 | 4 |

*Tabla 127. Carga de trabajo del escenario aleatorio*

Como se puede observar, no existe ningún tipo de criterio para la selección de la carga de trabajo. Se tiene un total de treinta aplicaciones.

La Tabla 128 y la Ilustración 48 presentan los resultados obtenidos.

| Tiempo secuencial (s) | Tiempo fuera de orden (s) | Tiempo con maleabilidad (s) | Tiempo con sobresuscripción (s) |
|---|---|---|---|
| 11870.37 | 11430.30 | 11200.56 | 11855.33 |

*Tabla 128. Tiempos de planificación para el escenario aleatorio*

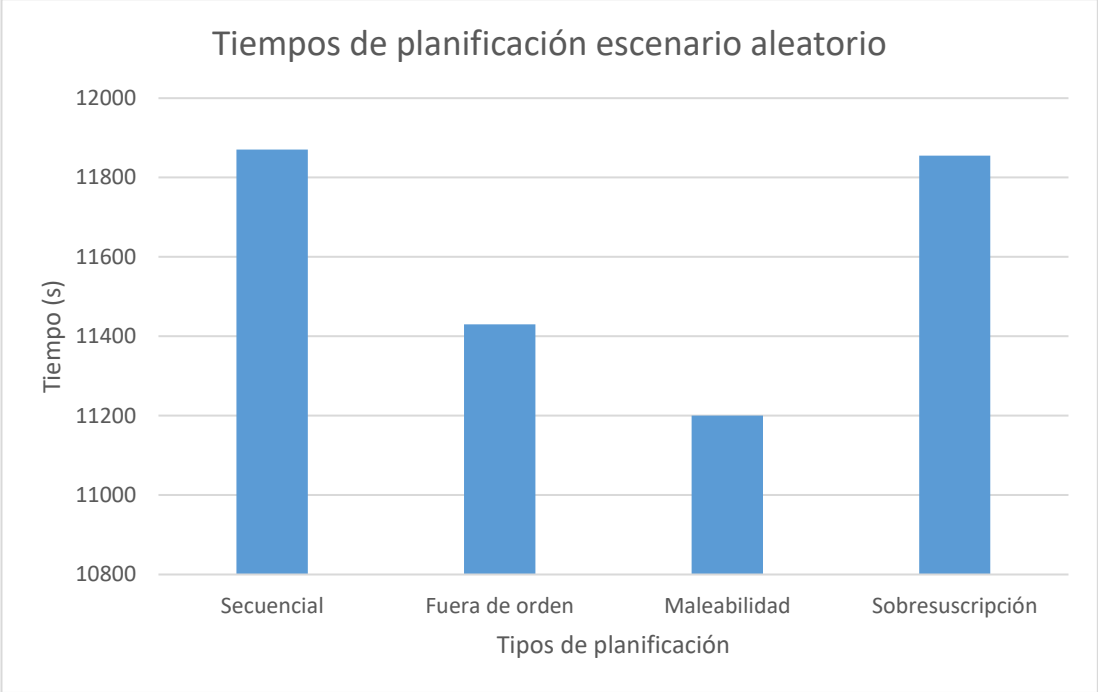



*Ilustración 48. Tiempos de planificación para el escenario aleatorio*

Ante los resultados obtenidos se puede concluir que, aunque la planificación secuencial respeta el orden, y por tanto antigüedad, de las aplicaciones, su aplicación se deriva en el peor tiempo obtenido, ya que los recursos quedan desaprovechados mayoritariamente.

Permitiendo la ejecución fuera de orden se mejora significativamente el tiempo de ejecución del sistema, ahorrando, en el orden de minutos de tiempo. Además, la ejecución de la maleabilidad mejora todavía más este tiempo, por lo que se puede afirmar que su aplicación ha resultado beneficiosa en este escenario.

Por su parte, aunque la planificación con sobresuscripción es capaz de mejorar la versión secuencial, no supone un buen tiempo respecto a las dos planificaciones anteriores, conociendo que no se han producido las mejores combinaciones de sobresuscripción.

Capítulo 6

# Conclusiones y trabajo futuro

Este capítulo finaliza el documento exponiendo las conclusiones obtenidas tras la realización del proyecto. Además, se presentan distintas líneas de trabajo e investigación que se pueden realizar en el futuro, con el objetivo de seguir ampliando las funcionalidades de la biblioteca FlexMPI y desarrollar más mecanismos de gestión central de las aplicaciones paralelas presentes en un sistema de computación de alto rendimiento.

## 6.1 Conclusiones

Durante la realización del presente proyecto se ha efectuado el diseño e implementación de mecanismos que permitan gestionar las aplicaciones paralelas de un sistema computacional de alto rendimiento. Para conseguir los objetivos propuestos, se han desarrollado nuevas funcionalidades para la herramienta FlexMPI, se ha desarrollado un componente controlador capaz de monitorizar y planificar las aplicaciones de un sistema de computación de alto rendimiento, se ha analizado y desplegado FlexMPI y las bibliotecas externas utilizadas junto con aplicaciones paralelas basadas en *benchmarks* de cálculo matricial, se han desarrollado técnicas de mapeo, mecanismos de planificación, modelos de maleabilidad y sobresuscripción y se han evaluado los resultados obtenidos.

A través de una infraestructura de control central, que es capaz de comunicarse con las aplicaciones paralelas, se monitorizan los datos de rendimiento que permiten conocer el estado de las aplicaciones y se decide su orden de ejecución conforme a los objetivos que se desean conseguir. La arquitectura de este controlador está diseñada de manera que se pueda seguir progresando en el futuro con nuevas funcionalidades y mecanismos de gestión.

Una de las políticas que permiten decidir la forma en que las aplicaciones del sistema se ejecutan es la planificación secuencial. Dicha política se fundamenta en el hecho de respetar el orden de llegada de las aplicaciones en el sistema para su ejecución. De esta manera, ejecuta estrictamente por orden las aplicaciones en función de los núcleos de procesamiento libres del sistema, deteniendo la planificación si no hay recursos disponibles y esperando hasta el momento en el que dichos recursos sean liberados. A pesar de que este tipo de planificación puede resultar justo desde el punto de vista de la antigüedad y tiempo de espera de las aplicaciones, no resulta demasiado eficiente, pues a menudo desperdicia recursos que podrían aprovechar otras aplicaciones que también esperan por su ejecución.

De esta manera, se considera la política de planificación fuera de orden, que no respeta el orden definido en la carga de trabajo y permite encontrar aplicaciones que asignar en núcleos de procesamiento no aprovechados. Se puede llegar a la conclusión claramente de que este tipo de política mejora significativamente los tiempos de ejecución del sistema de aplicaciones completo.

Sin embargo, en muchos casos también quedan núcleos no utilizados, situación que pretende corregir la planificación fuera de orden con maleabilidad. Este tipo de planificación aumenta el número de procesos si hay recursos libres y no hay ninguna otra aplicación que exija un número de procesos que le permita ser ejecutada. Se puede concluir que es una política muy adecuada y que asegura aprovechar todos los recursos del sistema en cada momento. Sin embargo, también puede presentar escenarios extremos en los que las configuraciones que adopta no son las más adecuadas, ya que, al incrementar una aplicación en el momento actual, puede derivar en que otra aplicación disponible para ejecutar en un tiempo reciente no pueda comenzar y deba esperar a que la aplicación maleabilizada termine su ejecución.

La última política considerada supone una planificación muy interesante que intenta aprovechar los recursos desde otra aproximación. Se fundamenta en el hecho de que mientras una aplicación presenta periodos de actividad, como por ejemplo la espera de un dato accedido en memoria, otra aplicación puede aprovechar dichos ciclos en el que el núcleo de procesamiento está ocioso para poder realizar su procesamiento activo. Sin embargo, resulta una planificación delicada que puede empeorar los tiempos de ejecución del sistema completo en diversos escenarios en los que los beneficios de realizar sobresuscripción no cubren las penalizaciones derivadas de los cambios de contexto de las aplicaciones para compartir el núcleo de procesamiento. Se puede deducir de las pruebas realizadas que resulta un buen tipo de política de planificación para aquellos casos en los que el objetivo no sea que cada aplicación se ejecute en el menor tiempo posible, porque de hecho su ejecución individual será más lenta,

sino que la meta sea aumentar el volumen de trabajo por unidad de tiempo que realiza el procesador, pues el tiempo completo de ejecución sí que será más rápido.

Desde el punto de vista del estudiante, se puede asegurar que se han alcanzado los objetivos que se perseguían al comienzo del desarrollo del proyecto. Para su realización, han sido fundamentales los conocimientos adquiridos en la mención de computadores, en la que se profundiza sobre aspectos del desarrollo de software de sistemas y sistemas operativos.

Cabe destacar, las numerosas dificultades surgidas para realizar el despliegue del sistema en el *cluster* Tucán de la Universidad Carlos III de Madrid que requería conocimientos avanzados de la arquitectura. Además, hubo periodos de tiempo en el que se presentaban problemas de disponibilidad que atrasaban el desarrollo del proyecto.

Además, al ser un proyecto principalmente de investigación, no hay unas referencias claramente definidas que faciliten su construcción, y cada pequeño paso suponía un gran esfuerzo.

Para concluir, se afirma que el desarrollo del trabajo ha resultado muy satisfactorio e interesante, y durante su realización se han aprendido muchos aspectos acerca del ámbito estudiado y sobre la realización de proyectos *software*.

## 6.2 Trabajo futuro

El propósito de este apartado es motivar el desarrollo de nuevos trabajos que pueden continuar el desarrollo del presente proyecto y anteriores estudios en los que se apoya.

En primer lugar, cabe destacar que el alumno seguirá colaborando en el desarrollo de nuevas funcionalidades para FlexMPI, como, por ejemplo, la integración del presente trabajo con otro proyecto fin de grado que desarrolla una interfaz gráfica para controlar las funcionalidades que ofrece FlexMPI. El objetivo de esta integración es proporcionar una visualización gráfica de las métricas de rendimiento que recibe el controlador y aplicar las acciones que éste permite de manera interactiva.

Por otra parte, aunque el sistema desarrollado está diseñado para establecer sus funcionalidades sobre varios nodos de un mismo sistema, no suponía el objetivo del presente proyecto y, por tanto, esta acción forma parte del trabajo futuro. El tutor y el alumno pretenden ajustar estas nuevas actividades, derivando en la posibilidad de llevar a cabo una publicación científica.

Como una nueva contribución a los modelos de caracterización diseñados, se propone el desarrollo de un modelo capaz de tener en cuenta el comportamiento de una aplicación dinámicamente, ya que normalmente no se comporta de una manera constante, sino que intercala fases de distinto tipo a lo largo del tiempo. La clave de este modelado se basa en extraer información interna de la propia aplicación, acción que ya realiza el controlador desarrollado a través de los contadores *hardware*.

También se plantea la realización de un modelo de aprendizaje y predicción, que, en base a experiencias previas, tenga la habilidad de determinar una planificación eficiente teniendo en cuenta los recursos y las aplicaciones existentes en el sistema.

Teniendo en cuenta la herramienta FlexMPI, la cual trabaja el paralelismo desde el enfoque de procesos de sistema, se proponen el desarrollo de una funcionalidad que permita aplicar un modelo híbrido con hilos. Este modelo puede resultar beneficioso en numerosos escenarios, debido a que proporcionan menores necesidades de rutinas comunicación y consumición de memoria. Además, se plantea un trabajo de investigación basado en exportar la lógica que aplica FlexMPI a sistemas de entrada/salida paralelos, proporcionando mecanismos de paralelismo a aplicaciones intensivas en este tipo de operaciones.

Anexo A

# Acrónimos

- **API** Application Programming Interface
- **BSD** Berkeley Software Distribution
- **CG** Conjugate Gradient
- **CPU** Central Processing Unit
- **EGS** Equi-Grow & Shrink
- **FLOPS** Floating Point Operations Per Second
- **FPSMA** Favor Previously Started Malleable Applications
- **GCC** GNU Compiler Collection
- **GNU** GNU is Not Unix
- **GPL** General Public License
- **HPC** High-Performance Computing
- **IEEE** Institute of Electrical and Electronics Engineers
- **IVA** Impuesto de valor agregado
- **KOJAK** Kit for Objective Judgement and Knowledge-based Detection of Performance Bottlenecks
- **LOPD** Ley Orgánica de Protección de Datos
- **LTS** Long Term Support
- **MPI** Message Passing Interface
- **MRI** Monitoring Request Interface
- **NAS** NASA Advanced Supercomputing
- **NASA** National Aeronautics and Space Administration

- **NPB** NAS Parallel Benchmarks
- **OpenMP** Open MultiProcessor
- **PAPI** Performance Application Programming Interface
- **PCM** Process Checkpointing and Migration
- **SCR** Scalable Checkpoint Restart
- **SFTP** SSH File Transfer Protocol
- **SRS** Software Requirements Specification
- **SSH** Secure Shell
- **TAU** Tuning and Analysis Utilities
- **ULFM** User Level Failure Mitigation

# Anexo B

# Manual de instalación

La herramienta FlexMPI se distribuye como un archive con extensión *tar.gz* y por tanto debe ser comprimida. Además, se debe realizar la integración de las bibliotecas externas que utiliza.

Los pasos que se deben seguir para realizar la instalación de **MPICH** de manera resumida son:

1. Obtener la distribución de MPICH y descomprimirla
    a. tar xzf mpich-XXXX.tar.gz

2. Realizar la instalación sobre la carpeta deseada
    a. ./mpich-XXXX/configure --prefix="Directorio escogido"

3. Construir MPICH sobre el directorio
    a. make

4. Instalar el paquete de MPICH
    a. make install

5. Añadir el directorio /bin generado al a la variable de entorno *path* del sistema
    a. PATH=$PATH:/"Directorio escogido"/bin

Los pasos requeridos para instalar integrar **PAPI** en el sistema de manera resumida son:

1. Obtener la distribución de PAPI y descomprimirla
    a. tar xzf papi-XXXX.tar.gz

2. Realizar la instalación sobre la carpeta deseada
    a. ./papi-XXXX/configure --prefix="Directorio escogido"

3. Construir PAPI sobre el directorio
    a. make

4. Instalar el paquete de PAPI
    a. make install

Para la integración de **FlexMPI,** en su versión con las modificaciones añadidas en este proyecto, en el sistema se deben realizar las siguientes operaciones:

1. Descomprimir la herramienta
    a. tar xzf flexmpi.tar.gz

2. Modificar los valores de las constantes LIBEMPI_DIR, MPICC y PAPI_DIR en flexmpi/Makefile por los directorios escogidos de las biblitecas mencionadas.

3. Construir FlexMPI
    a. make

4. Integrar FlexMPI en la aplicación paralela deseada en función de la lógica y llamadas MPI que realice.

Para integrar el **controlador** desarrollado:

1. Acceder al directorio FlexMPI/API y descomprimir el archivo controller-XXXX.tar.gz
    a. cd FlexMPI/API
    b. tar xzf controller-XXXX.tar.gz

2. Modificar el fichero workload para definir la carga de trabajo:
    a. Abrir el fichero con un editor de texto
    b. Para cada nueva aplicación escribir una línea con el formato: tipo:número_iteraciones:nodo:número_procesos

3. Construir el controlador
    a. make

4. Ejecutar el controlador
    a. ./controller

El controlador se compone de los siguientes ficheros:

- **controller.c**. Fichero que contiene la implementación de las funciones involucradas.

- **constants.h.** Fichero que reúne todas las constantes, parámetros y mensajes.

- **system_includes.h.** Fichero que incluye las importaciones del sistema.

- **structs**.h. Fichero que define las estructuras utilizadas.

- **functions_declarations.h**. Fichero que agrupa la declaración de las funciones utilizadas.

- **doubly_linked_list.h**. Fichero que define las estructuras y funciones que involucran a las listas doblemente enlazadas utilizadas.

- **doubly_linked_list.c**. Fichero que contiene la implementación de las funciones con las que trabajan las listas doblemente enlazadas.

# Anexo C

# Manual de usuario

El presente manual tiene el objetivo de explicar el flujo de ejecución de las funcionalidades contribuidas por el proyecto descrito. Se asume que se ha completado la instalación de los distintos componentes y se ha realizado el correcto despliegue de las herramientas descritas.

El flujo principal de ejecución se establece al ejecutar el fichero ejecutable del controlador obtenido durante la compilación.

Una vez iniciada, la aplicación mostrará por salida estándar toda la información referente a la inicialización, incluyendo la carga de trabajo definida.

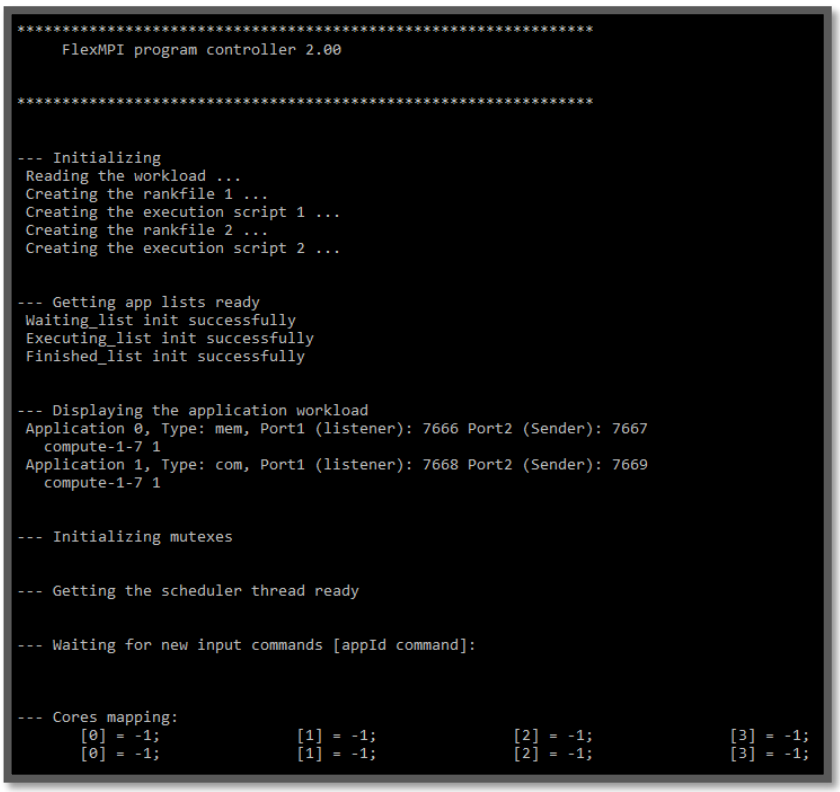



*Ilustración 49. Inicio del controlador*

A partir de este estado, se podrán introducir los comandos de la aplicación. El comando de selección de política de planificación se compone del prefijo *policy_* acompañado de la política a seleccionar. Por tanto, el formato de los comandos es:

- policy_batch: planificación secuencial
- policy_ooo: planificación fuera de orden
- policy_ooomalleability: planificación fuera de orden con maleabilidad
- policy_oversubscription: planificación con sobresuscripción

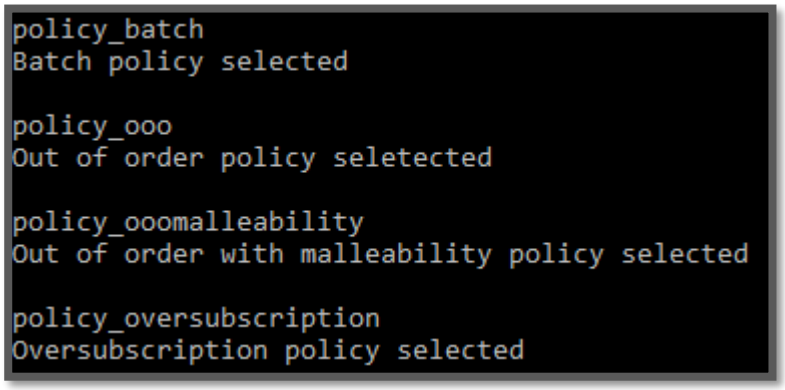



*Ilustración 50. Selección de la política de planificación*

El comando para arrancar el sistema de planificación permite comenzar la planificación bajo la política que se haya escogido en ese momento. Si no se especifica ninguna política, por defecto se comienza con la política de planificación secuencial. Cabe destacar, que, si el sistema estaba arrancado y se encuentra en marcha, se ignora este comando al introducirlo. El formato del comando es:

- start: permite comenzar la planificación del sistema

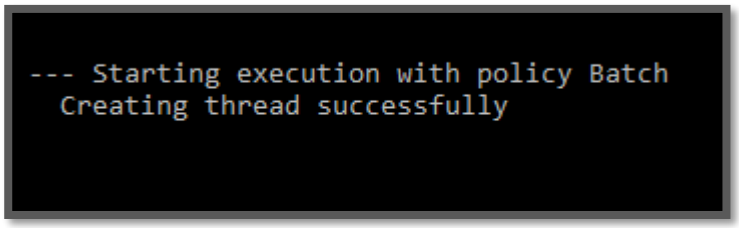



*Ilustración 51. Comienzo de la planificación*

El comando para suspender el sistema de planificación permite pausar la planificación vigente. El comando es:

- stop: permite pausar el sistema de planificación


*Ilustración 52. Suspensión de la planificación*

Por su parte, el comando para reanudar la planificación si estaba suspendida es el mismo que se utiliza para el comienzo del sistema. El formato del comando es:

- start: permite reanudar la planificación si ésta se encontraba suspendida

El comando de finalización permite terminar con la ejecución del sistema completo, aunque se encuentre en ejecución. Antes de terminar, se indica a las aplicaciones paralelas que finalicen su ejecución. El formato del comando es:

- exit: finaliza la ejecución del sistema completo, aunque se encuentre en marcha


*Ilustración 53. Finalización de la ejecución*

Cuando una aplicación inicia su ejecución se informa de este hecho, mostrando el mapeo que realizará y el comienzo de la monitorización y su mapeo remoto.

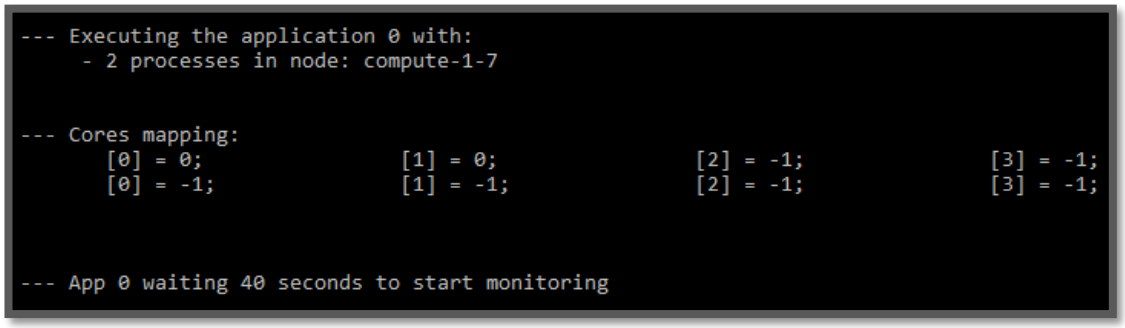



*Ilustración 54. Inicio de la ejecución de una aplicación*

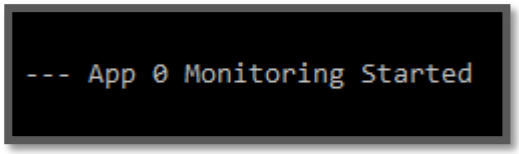



*Ilustración 55. Inicio de la monitorización de una aplicación*

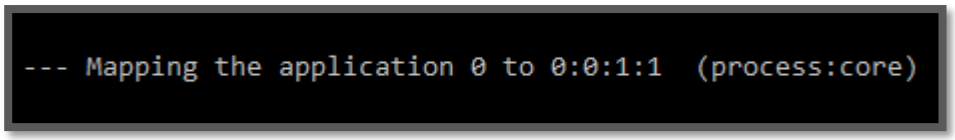



*Ilustración 56. Instrucción de asignación de procesos y núcleos de una aplicación*

Cuando una aplicación finaliza se informa del evento y se muestra la asignación de núcleos de procesamiento tras su liberación.

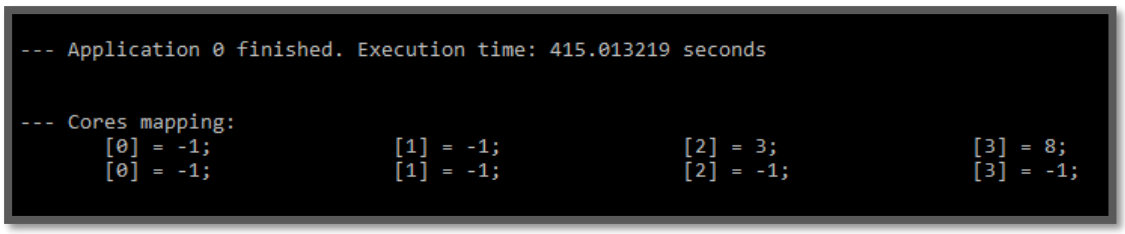



*Ilustración 57. Finalización de una aplicación*

Cuando una aplicación aumenta el número de procesos, debido a la aplicación de maleabilidad, se informa del evento y se muestra la asignación de núcleos tras el incremento.

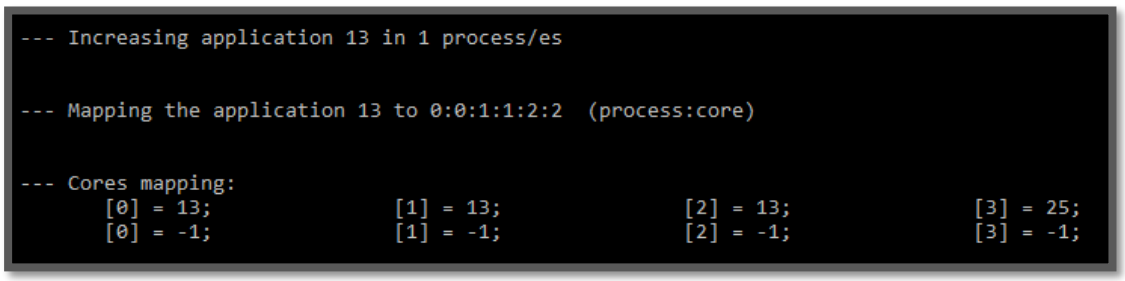



*Ilustración 58. Maleabilidad de una aplicación*

Cuando una aplicación se sobresuscribe en un núcleo de procesamiento se muestra el mapeo que realizará y el comienzo de la monitorización y su mapeo remoto.

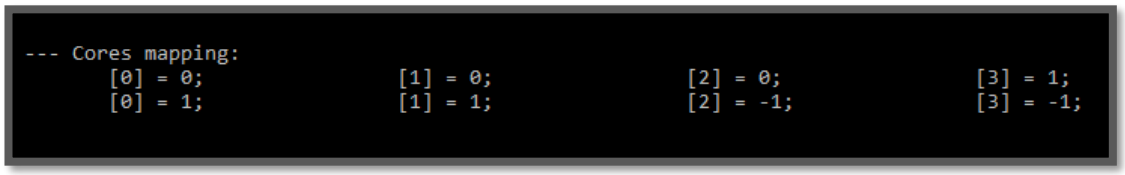



*Ilustración 59. Sobresuscripción de procesos*

Cuando una aplicación encuentra una mejor configuración de sobresuscripción muestra el nuevo mapeo que realizará.

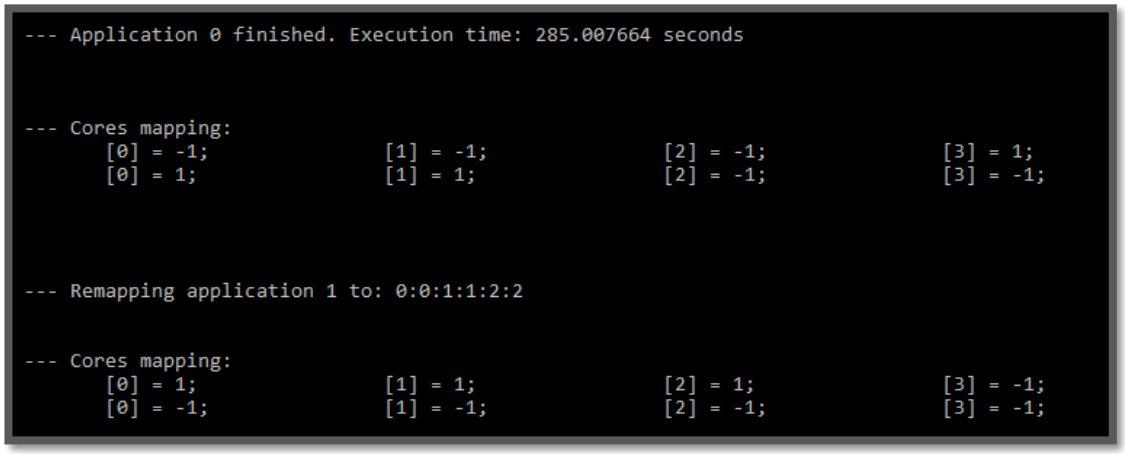



*Ilustración 60. Nuevo mapeo tras encontrar una mejor configuración de sobresuscripción*

# Bibliografía

[1] BERGMAN, Keren, et al. Exascale computing study: Technology challenges in achieving exascale systems. Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep, 2008, vol. 15.

[2] NATIONAL RESEARCH COUNCIL, et al. The potential impact of high-end capability computing on four illustrative fields of science and engineering. National Academies Press, 2008.

[3] KISH, Laszlo B. End of Moore's law: thermal (noise) death of integration in micro and nano electronics. Physics Letters A, 2002, vol. 305, no 3, p. 144-149.

[4] SHALF, John, et al. Exascale computing technology challenges. International Conference on High Performance Computing for Computational Science. Springer Berlin Heidelberg, 2010. p. 1-25.

[5] BERGMAN, Keren, et al. Exascale computing study: Technology challenges in achieving exascale systems. Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep, 2008, vol. 15.

[6] MAGHRAOUI, K. E., et al. Dynamic malleability in iterative MPI applications. En 7th Int. Symposium on Cluster Computing and the Grid. 2007. p. 591-598.

[7] SNIR, Marc. MPI--the Complete Reference: The MPI core. MIT press, 1998.

[8] MARTÍN, Gonzalo, et al. Enhancing the performance of malleable MPI applications by using performance-aware dynamic reconfiguration. Parallel Computing, 2015, vol. 46, p. 60-77.

[9] VADHIYAR, Sathish S.; DONGARRA, Jack J. Srs: A framework for developing malleable and migratable parallel applications for distributed systems. Parallel Processing Letters, 2003, vol. 13, no 02, p. 291-312.

[10] MARCIA C. C., et al. Supporting Malleability in Parallel Architectures with Dynamic CPUSET Mapping and Dynamic MPI. Lecture Notes in Computer Science, 2010, vol. 5935, p. 242-257.

[11] SUDARSAN, Rajesh; RIBBENS, Calvin J. ReSHAPE: A framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment. Parallel Processing, 2007. ICPP 2007. International Conference on. IEEE, 2007. p. 44-44.

[12] UTRERA, Gladys; CORBALAN, Julita; LABARTA, Jesus. Implementing malleability on MPI jobs. Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques. IEEE Computer Society, 2004. p. 215-224.

[13] EL MAGHRAOUI, Kaoutar; SZYMANSKI, Boleslaw K.; VARELA, Carlos. An architecture for reconfigurable iterative MPI applications in dynamic environments. International Conference on Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, 2005. p. 258-271.

[14] EL MAGHRAOUI, Kaoutar, et al. Malleable iterative MPI applications. Concurrency and Computation: Practice and Experience, 2009, vol. 21, no 3, p. 393-413.

[15] PRABHAKARAN, Suraj, et al. A batch system with fair scheduling for evolving applications. Parallel Processing (ICPP), 2014 43rd International Conference on. IEEE, 2014. p. 351-360.

[16] PRABHAKARAN, Suraj, et al. A batch system with efficient adaptive scheduling for malleable and evolving applications. Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International. IEEE, 2015. p. 429-438.

[17] LEMARINIER, Pierre, et al. Architecting Malleable MPI Applications for Priority-driven Adaptive Scheduling. Proceedings of the 23rd European MPI Users' Group Meeting. ACM, 2016. p. 74-81.

[18] SUDARSAN, Rajesh; RIBBENS, Calvin J. Scheduling resizable parallel applications. Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE, 2009. p. 1-10.

[19] UTRERA, Gladys, et al. A job scheduling approach for multi-core clusters based on virtual malleability. En European Conference on Parallel Processing. Springer Berlin Heidelberg, 2012. p. 191-203.

[20] SUDARSAN, Rajesh; RIBBENS, Calvin J. Combining performance and priority for scheduling resizable parallel applications. Journal of Parallel and Distributed Computing, 2016, vol. 87, p. 55-66.

[21] SONMEZ, Ozan, et al. Scheduling malleable applications in multicluster systems. Cluster Computing, 2007 IEEE International Conference on. IEEE, 2007. p. 372-381.

[22] CARTER, John B., et al. Shared memory computer networks. U.S. Patent No 6,148,377, 14 Nov. 2000.

[23] HADDOCK, Stephen R., et al. Distributed memory switching hub. U.S. Patent No 6,175,571, 16 Ene. 2001.

[24] DAGUM, Leonardo; MENON, Ramesh. OpenMP: an industry standard API for shared-memory programming. IEEE computational science and engineering, 1998, vol. 5, no 1, p. 46-55.

[25] CHORLEY, Martin James. Performance Comparison of Message Passing and Shared Memory Programming with HPC Benchmarks. 2007. Tesis Doctoral. master thesis, The University of Edinburgh.

[26] LUSK, Ewing; CHAN, Anthony. Early experiments with the OpenMP/MPI hybrid programming model. International Workshop on OpenMP. Springer Berlin Heidelberg, 2008. p. 36-47.

[27] BASUMALLIK, Ayon; EIGENMANN, Rudolf. Towards automatic translation of OpenMP to MPI. Proceedings of the 19th annual international conference on Supercomputing. ACM, 2005. p. 189-198.

[28] MALONY, Allen D. Tools for parallel computing: A performance evaluation perspective. Handbook on Parallel and Distributed Processing. Springer Berlin Heidelberg, 2000. p. 342-363.

[29] KNÜPFER, Andreas, et al. The vampir performance analysis tool-set. Tools for High Performance Computing. Springer Berlin Heidelberg, 2008. p. 139-155.

[30] SHENDE, Sameer S.; MALONY, Allen D. The TAU parallel performance system. The International Journal of High Performance Computing Applications, 2006, vol. 20, no 2, p. 287-311.

[31] MOHR, Bernd; WOLF, Felix. KOJAK–A tool set for automatic performance analysis of parallel programs. European Conference on Parallel Processing. Springer Berlin Heidelberg, 2003. p. 1301-1304.

[32] GEIMER, Markus, et al. Scalable Collation and Presentation of Call-Path Profile Data with CUBE. PARCO. 2007. p. 645-652.

[33] MILLER, Barton P., et al. The Paradyn parallel performance measurement tool. Computer, 1995, vol. 28, no 11, p. 37-46.

[34] GEIMER, Markus, et al. The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience, 2010, vol. 22, no 6, p. 702-719.

[35] MUCCI, Philip J., et al. PAPI: A portable interface to hardware performance counters. Proceedings of the department of defense HPCMP users group conference. 1999.

[36] SHEWCHUK, Jonathan Richard, et al. An introduction to the conjugate gradient method without the agonizing pain. 1994.

[37] FORSYTHE, George Elmer; HENRICI, Peter. The cyclic Jacobi method for computing the principal values of a complex matrix. Transactions of the American Mathematical Society, 1960, vol. 94, no 1, p. 1-23.

[38] STACK EXCHANGE INC. 2016, Developer Survey results 2016. [online]. 2016. [Accessed 08 June 2016]. Available from: https://insights.stackoverflow.com/survey/2016

[39] BALAJI, S.; MURUGAIYAN, M. Sundararajan. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. International Journal of Information Technology and Business Management, 2012, vol. 2, no 1, p. 26-30.

[40] IEEE COMPUTER SOCIETY. SOFTWARE ENGINEERING STANDARDS COMMITTEE; IEEE-SA STANDARDS BOARD. IEEE recommended practice for software requirements specifications.

[41] MARTÍN CRUZ, Gonzalo. Optimization techniques for adaptability in MPI application. (Doctoral thesis). 2015.

[42] MPICH.ORG. 2017, MPICH downloads. [online]. 2017. [Accessed 09 June 2017]. Available from: http://www.mpich.org/downloads/

[43] INNOVATIVE COMPUTING LABORATORY, UNIVERSITY OF TENNESSE. 2017, PAPI software. [online]. 2017. [Accessed 09 June 2017]. Available from: http://icl.cs.utk.edu/papi/software/#license

[44] UNIVERSITY CARLOS III DE MADRID. 2017, Tucán [online]. 2017. [Accessed 09 June 2017]. Available from: http://tucan.arcos.inf.uc3m.es/